Implementasi Algoritme Kriptografi Lizard untuk Mengamankan Pengiriman Data Menggunakan Arsitektur *Web Service* REST pada Mikrokontroler NodeMCU

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Kalbuadi Joyoputro
NIM: 145150200111028

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

Implementasi Algoritme Kriptografi Lizard
untuk Mengamankan Pengiriman Data
Menggunakan Arsitektur Web Service REST
pada Mikrokontroler NodeMCU

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Kalbuadi Joyoputro
NIM: 145150200111028

Skripsi ini telah diuji dan dinyatakan lulus pada
26 Juli 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Ari Kusyanti, S.T., M.Sc.	Fariz Andri Bakhtiar, S.T., M.Kom.
NIK: 201102 831228 2 001	NIK: 201709 840314 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D
NIP: 19710518 200312 1 001

ii

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata di dalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 26 Juli 2018

Kalbuadi Joyoputro

NIM: 145150200111028

iii

ABSTRAK

Kalbuadi Joyoputro, Implementasi Algoritme Kriptografi Lizard untuk Mengamankan Pengiriman Data Menggunakan Arsitektur *Web Service* REST pada Mikrokontroler NodeMCU

Dosen Pembimbing: Ari Kusyanti, S.T, M.Sc dan Fariz Andri Bakhtiar, S.T, M. Kom

Keamanan merupakan salah satu aspek penting yang sering terlupakan dalam hal transaksi data antarperangkat *Internet of Things* (IoT) seperti mikrokontroler. Ada tiga unsur dalam mekanisme keamanan *CIA* (*Confidentiality*, *Integrity*, *Availability*), yaitu kerahasiaan, integritas, dan ketersediaan data. Penelitian ini berfokus pada bagaimana cara memenuhi unsur kerahasiaan data yang dikirimkan oleh perangkat mikrokontroler NodeMCU dalam suatu sistem IoT. Memori dinamis dan prosesor perangkat NodeMCU yang terbilang kecil menjadi suatu tantangan tersendiri dalam memenuhi unsur kerahasiaan data. Salah satu cara untuk memenuhi unsur kerahasiaan data dan menjawab tantangan tersebut adalah dengan mengimplementasikan algoritme kriptografi Lizard pada perangkat NodeMCU. Data berbentuk JSON (*JavaScript Object Notation*) akan dienkripsi menjadi *cipher text* menggunakan *keystream* algoritme Lizard sebelum dikirim menuju server basis data melalui jaringan *wifi*. Pada server, data akan didekripsi menjadi bentuk semula dan disimpan ke dalam basis data. Proses pengiriman data menggunakan arsitektur *web service* REST (*Representational State Transfer*). Terdapat empat bagian pengujian untuk memastikan sistem telah berjalan dengan baik yaitu pengujian validasi *test vector*, pengujian fungsional, pengujian kinerja waktu dan memori, serta pengujian keamanan. Algoritme kriptografi Lizard dapat diimplementasikan dan berjalan dengan baik pada perangkat mikrokontroler dengan daya yang terbatas seperti NodeMCU. Hasil pengujian kinerja menunjukkan bahwa pembentukan 216 dan 352 bit keystream pada perangkat NodeMCU membutuhkan waktu 0,01 detik dan memori sebesar 3,4% dari memori total. Sedangkan proses enkripsi 216 dan 352 bit plain text membutuhkan waktu 0,02 dan 0,04 detik serta memori sebesar 0,3% dari memori total. Pengimplementasian algoritme Lizard berhasil memenuhi unsur kerahasiaan pada proses pengiriman data.

Kata kunci: kriptografi, algoritme Lizard, IoT, NodeMCU, *web service* REST

ABSTRACT

Kalbuadi Joyoputro, ***Implementation of Lizard Algorithm for Securing Data Delivery Using*** **REST** ***Web Service Architecture on NodeMCU Microcontroller***

Supervisors**: Ari Kusyanti, S.T, M.Sc** ***and*** **Fariz Andri Bakhtiar, S.T, M. Kom**

Security is one important aspect that is often forgotten in terms of data transactions between IoT (Internet of Things) devices such as microcontrollers. There are three elements in CIA (Confidentiality, Integrity, and Availability) triad security namely data confidentiality, integrity, and availability. This research focus on how to complete the data confidentiality on NodeMCU microcontroller device in an IoT system. Relatively small NodeMCU processor and dynamic memory becomes challenge in fulfilling the data confidentiality. One way to fulfill the element and answer the challenge is to implement the Lizard stream cipher algorithm on NodeMCU devices. The JSON (JavaScript Object Notation) data will be encrypted into cipher text using the Lizard keystream algorithm before being sent to the database server over the wireless wifi network. On the server side, the data will be decrypted into its original form and stored into the database. The process of sending data uses the REST (Representational State Transfer) web service architecture. There are four parts of testing to ensure the system has been running well that is test vector validation testing, functional testing, performance and memory performance testing, and security testing. Lizard cryptographic algorithm can be implemented and running well on microcontroller devices with limited power such as NodeMCU. The 216 and 352 bit keystream generated on NodeMCU devices takes 0.01 seconds and 3.4% of total dynamic memory, while encrypting 216 and 352 bits plain text takes 0.02 and 0.04 seconds with 0.3% of total dynamic memory. Lizard cryptographic algorithm can be implemented and running well on microcontroller devices with limited power such as NodeMCU. Implementation of Lizard algorithm has successfully guaranteed the data confidentiality.

Keywords*: cryptography, Lizard algorithm, IoT, NodeMCU, REST web service*

KATA PENGANTAR

Puji syukur ke hadirat Tuhan Yang Maha Esa atas berkat dan penyertaan-Nya sehingga penulis dapat menyelesaikan penulisan skripsi berjudul "Implementasi Algoritme Kriptografi Lizard untuk Mengamankan Pengiriman Data Menggunakan Arsitektur Web Service REST pada Mikrokontroler NodeMCU" dengan baik. Penulisan skripsi ini diajukan untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer pada Proram Studi Teknik Informatika Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.

Penulis menyadari bahwa skripsi ini tidak akan berhasil tanpa bantuan dari beberapa pihak yang telah memberikan bantuan baik lahir maupun batin selama penulisan skripsi ini. Oleh karena itu, penulis ingin menyampaikan rasa hormat dan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Orang tua yang selalu mendukung, mengerti, dan berdoa bagi penulis agar dapat menyelesaikan penelitian dan penulisan skripsi ini,
2. Ibu Ari Kusyanti, S.T, M.Sc dan Bapak Fariz Andri Bakhtiar, S.T., M.Kom selaku dosen pembimbing I dan pembimbing II yang telah dengan sabar membimbing, mengarahkan, serta meluangkan waktu untuk penulis sehingga dapat menyelesaikan skripsi ini,
3. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D, Bapak Agus Wahyu Widodo, S.T, M.Sc dan Bapak M. Tanzil Furqon, S.Kom, M.CompSc selaku Ketua Jurusan Teknik Informatika, Ketua Program Studi Teknik Informatika dan Sekretaris jurusan Teknik Informatika,
4. Bapak Achmad Basuki, S.T, M.MG, Ph.D selaku Dosen Pembimbing Akademik yang telah mengarahkan serta memantau kegiatan akademik penulis,
5. Seluruh dosen dan civitas Program Studi Informatika Universitas Brawijaya atas kesediaan membagi ilmunya kepada penulis,
6. Teman-teman kontrakan kalpataru yang selalu bersedia membantu dan mendukung penulis,
7. Keluarga Advokesma BEM FILKOM 2016 atas kerjasama dan pengabdian selama satu periode hingga rasa kekeluargaan yang terus berlanjut hingga sekarang,
8. Teman-teman angkatan 2014 serta para senior dan junior atas segala bantuan dan kenangan selama menempuh studi di Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya,
9. Oma, Opa, Budhe, Pakdhe, dan kakak-kakak sepupu yang telah membantu dan mendukung jasmani serta rohani penulis,
10. Perempuan-perempuan yang pernah menjadi kekasih dan tetap menjadi teman dekat atas semangat dan dukungan hingga sekarang,
11. Dan seluruh pihak yang telah membantu kelancaran penulisan skripsi yang tidak dapat penulis sebutkan satu persatu.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak kekurangan, sehingga saran dan kritik yang membangun senantiasa penulis harapkan. Semoga skripsi ini dapat memberikan manfaat bagi semua pihak membacanya.

Malang, 26 Juli 2018

Penulis

kalbu_joyo@student.ub.ac.id

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

Bab pendahuluan ini terdiri dari latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah, dan sistematika pembahasan penelitian.

1.1 Latar Belakang

Perkembangan teknologi yang pesat mengakibatkan dunia memasuki era digitalisasi data. Manusia berusaha mendapatkan data dari lingkungan dan objek di sekitarnya dengan bantuan teknologi. Teknologi tersebut biasa dikenal dengan sebutan *Internet of Things* (IoT). IoT merupakan teknologi komunikasi masa depan, di mana mikrokontroler akan ditanamkan pada setiap objek yang ada di lingkungan sekitar yang membuat objek tersebut dapat berkomunikasi dengan objek lainnya dan juga dengan penggunanya yaitu manusia (Sembiring, 2017). IoT memanfaatkan teknologi internet sebagai konektivitas dan pengiriman datanya sehingga dapat memberikan informasi secara *real-time* tanpa campur tangan manusia, atau sering disebut komunikasi *machine to machine*.

Jika melihat cara kerja IoT, data yang dikirim melalui media transmisi nirkabel sangat rentan untuk diserang. Akan menjadi lebih parah, apabila data yang dikirim masih berupa *plain text* yang dapat dibaca dengan mudah oleh sang penyerang. Jika terjadi pencurian data, maka unsur kerahasiaan telah gagal dalam sistem IoT tersebut. Untuk mengamankan data pada perangkat IoT, dilakukan penelitian guna menjaga kerahasiaan data tersebut dengan cara enkripsi. Berdasarkan standar *International Electrotechnical Commission* (*IEC*) 62591, enkripsi data pada sistem IoT merupakan salah satu unsur penting yang harus dipenuhi (Roman, Najera, & Lopez, 2011). Buku panduan *IEC* 62591 (2016) sendiri mengatakan bahwa setiap orang yang tidak memiliki wewenang seharusnya tidak dapat melihat dan membaca data digital dalam sebuah sistem jaringan nirkabel.

Penelitian tentang implementasi algoritme kriptografi untuk mengamankan data pada perangkat mikrokontroler berbasis Arduino telah dilakukan oleh Sembiring (2017) dengan mengimplementasikan algoritme kunci asimetri *Rivest–Shamir–Adleman* (*RSA*) untuk enkripsi data pada mikrokontroler Arduino Uno. Hasil uji coba menunjukkan bahwa proses enkripsi menggunakan algoritme kriptografi RSA dapat berjalan dengan baik walaupun memakan waktu dan memori yang tidak sedikit (Sembiring, 2017). Melihat hal tersebut, diperoleh gagasan untuk melakukan penelitian dengan menerapkan algoritme kriptografi yang ringan dan cocok diterapkan pada mikrokontroler berbasis Arduino. Mikrokontroler adalah salah satu perangkat keras yang digunakan sebagai *single board computer* bagi perangkat IoT (Rose, 2015).

Algoritme Lizard adalah sebuah algoritme kriptografi *stream cipher* yang tergolong ringan dan cocok digunakan pada perangkat yang memiliki keterbatasan daya (Hamann, 2017). Selain itu, algoritme Lizard pun belum pernah diimplementasikan pada perangkat mikrokontroler. Melihat hal tersebut, diperoleh gagasan untuk melakukan penelitian dengan menerapkan algoritme

kriptografi Lizard pada mikrokontroler berbasis Arduino. Terdapat berbagai macam mikrokontroler, salah satunya adalah NodeMCU. Penelitian ini menggunakan NodeMCU untuk membuktikan bahwa algoritme Lizard dapat berjalan pada perangkat yang memiliki keterbatasan daya dan memori yang kecil.

Algoritme Lizard diimplementasikan untuk melakukan enkripsi data pada NodeMCU sebelum dikirim menuju server basis data melalui jaringan *wifi* dengan menggunakan *web service*. Arsitektur *web service* REST (*Representational State Transfer*) sangat kompatibel untuk diimplementasikan pada sistem komunikasi *machine to machine* (Laine, 2011). Selain itu, arsitektur jenis ini merupakan standar *Internet Engineering Task Force (IETF)* yang dapat diterapkan pada teknologi IoT (Roman, Najera, & Lopez, 2011). Setelah dilakukan proses implementasi algoritme Lizard tersebut, akan dilakukan pengujian untuk menguji efektivitas dan efisiensi waktu proses serta penggunaan memori dari perangkat mikrokontroler yang digunakan. Penelitian implementasi algoritme kriptografi Lizard untuk mengamankan pengiriman data menggunakan arsitektur *web service* REST pada mikrokontroler NodeMCU diharapkan dapat menjadi langkah awal penerapan algoritme Lizard untuk proses enkripsi guna menjaga kerahasiaan data pada sistem berkonsep IoT.

1.2 Rumusan Masalah

Dengan latar belakang penelitian yang telah disebutkan sebelumnya, peneliti merumuskan beberapa permasalahan sebagai berikut:

1. Bagaimana implementasi algoritme kriptografi Lizard pada perangkat mikrokontroler NodeMCU untuk memenuhi unsur kerahasiaan saat pengiriman data?
2. Bagaimana validasi *keystream* sistem dengan *test vector* algoritme Lizard?
3. Bagaimana kinerja pembentukan *keystream* algoritme Lizard terhadap penggunaan waktu dan memori pada mikrokontroler NodeMCU?
4. Bagaimana kinerja pemrosesan enkripsi data menggunakan *keystream* algoritme Lizard terhadap penggunaan waktu dan memori pada mikrokontroler NodeMCU?

1.3 Tujuan

Dari masalah yang telah dirumuskan pada subbab 1.2, dapat ditentukan tujuan dari penelitian ini adalah sebagai berikut:

1. Melakukan implementasi algoritme kriptografi Lizard pada perangkat mikrokontroler NodeMCU untuk memenuhi unsur kerahasiaan saat pengiriman data.
2. Melakukan validasi *keystream* sistem dengan *test vector* algoritme Lizard.
3. Mengetahui kinerja pembentukan *keystream* algoritme Lizard pada mikrokontroler NodeMCU.
4. Mengetahui kinerja pemrosesan enkripsi data menggunakan *keystream* algoritme Lizard pada mikrokontroler NodeMCU.

1.4 Manfaat

Manfaat dari penelitian ini adalah untuk mengetahui kinerja algoritme Lizard pada perangkat mikrokontroler NodeMCU. Penelitian ini bermanfaaat untuk menjadikan algoritme Lizard sebagai pilihan dalam menjaga kerahasiaan data dalam suatu sistem yang menggunakan perangkat dengan daya terbatas, khususnya pada sistem IoT.

1.5 Batasan Masalah

Agar penyusunan skripsi ini sesuai dengan latar belakang dan selalu terfokus dengan rumusan masalah, maka batasan masalah pada penelitian ini adalah sebagai berikut:

1. Proses pengiriman data dari mikrokontroler NodeMCU menuju ke server basis data menggunakan arsitektur komunikasi *web service* REST secara nirkabel dengan jaringan Wi-Fi. Klien dan server berada pada jaringan lokal yang sama.
2. Enkripsi dilakukan pada mikrokontroler NodeMCU sebelum data dikirim menuju server basis data.
3. Data suhu dan kelembapan akan dikirim dalam bentuk JSON (*JavaScript Object Notation*).
4. Penelitian hanya berfokus pada unsur kerahasiaan data, sedangkan unsur integritas dan ketersediaan data tidak diperhitungkan.
5. *Keystream* algoritme Lizard hanya dibentuk pada mikrokontroler NodeMCU. *Keystream* pada sisi server diperoleh dari pembentukan di sisi klien dan dideklarasi secara manual pada sistem.

1.6 Sistematika Pembahasan

Sistematika pembahasan dari penyusunan penelitian yang direncanakan adalah sebagai berikut :

BAB I PENDAHULUAN

Bab pendahuluan terdiri dari latar belakang pembuatan skripsi, identifikasi, pembatasan masalah, rumusan masalah, tujuan dan manfaat, serta sistematika penulisan dari skripsi "Implementasi Algoritme Kriptografi Lizard untuk Mengamankan Pengiriman Data Suhu dan Kelembapan Menggunakan Arsitektur *Web Service* REST pada Mikrokontroler NodeMCU".

BAB II LANDASAN KEPUSTAKAAN

Bab landasan kepustakaan menguraikan tentang teori-teori yang dipakai dalam penelitian implementasi algoritme kriptografi Lizard untuk mengamankan pengiriman data suhu dan kelembapan menggunakan arsitektur *web service* REST pada mikrokontroler NodeMCU. Temuan dan bahan penelitian yang digunakan pada penelitian ini diperoleh dari berbagai referensi yang menunjang penelitian.

BAB III METODOLOGI

Bab metodologi ini menguraikan dan membahas langkah kerja dalam pengerjaan penelitian implementasi algoritme kriptografi Lizard untuk mengamankan pengiriman data menggunakan arsitektur *web service* REST pada mikrokontroler NodeMCU seperti studi literatur, perancangan dan implementasi, rekayasa kebutuhan sistem, pengujian, serta pembahasan hasil pengujian yang dilakukan.

BAB IV REKAYASA KEBUTUHAN

Bab rekayasa kebutuhan berisi tentang gambaran umum dari sistem yang mengimplementasikan algoritme kriptografi Lizard pada perangkat mikrokontroler NodeMCU. Selain itu bab ini juga berisi kebutuhan fungsional dan nonfungsional dari perangkat keras maupun perangkat lunak pembentuk sistem.

BAB V PERANCANGAN DAN IMPLEMENTASI

Bab perancangan dan implementasi menjelaskan bagaimana cara kerja sistem dan alur pengiriman data pada perangkat mikrokontroler NodeMCU dengan keamanan kriptografi Lizard. Implementasi menggambarkan tentang bagaimana sistem diterapkan berdasarkan perancangan yang telah dibuat sebelumnya secara rinci beserta langkah-langkah pengerjaan yang dilakukan, serta menampilkan gambar-gambar implementasi yang dilakukan.

BAB VI PENGUJIAN

Bab pengujian disusun guna menjelaskan kinerja algoritme kriptografi Lizard pada perangkat mikrokontroler NodeMCU. Bab ini juga menampilkan skenario pengujian yang ada pada sistem ini dan menyajikan data hasil pengujian dari implementasi sistem yang telah dibangun. Hasil pengujian diperoleh dari pengujian yang telah dilakukan sesuai dengan skenario yang telah ditentukan, dan dari hasil tersebut dapat diketahui kinerja sistem.

BAB VII PENUTUP

Pada bab penutup dipaparkan kesimpulan dari pelaksanaan penelitian secara keseluruhan. Kesimpulan ini disusun dari hasil pengujian terhadap kinerja algoritme kriptografi Lizard. Selain itu, saran dari penulis diberikan untuk pengembangan penelitian ini selanjutnya agar lebih baik dan disempurnakan.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab landasan kepustakaan ini terdapat tinjauan pustaka dari penelitian yang telah ada dan yang diusulkan. Bab landasan kepustakaan disusun dari berbagai sumber pustaka yang terkait dengan teori dan metode yang digunakan dalam penelitian ini. Adapun teori yang mendasari penelitian ini yaitu kriptografi, *Internet of Things*, mikrokontroler NodeMCU, algoritme Lizard, dan arsitektur komunikasi *web service* REST.

2.1 Kajian Pustaka

Paper berjudul "*LIZARD – A Lightweight Stream Cipher for Power-constrained Devices*" yang ditulis oleh Hamann, Krause, dan Meier menjadi rujukan utama dalam pengimplementasian algoritme Lizard. *Paper* tersebut berisi penjelasan tentang proses pembentukan *keystream* serta latar belakang diciptakannya algoritme Lizard. Pada penelitian tersebut, algoritme Lizard diimplementasikan pada perangkat keras *RFID*. Pada pengujian, algoritme Lizard tercatat menggunakan daya perangkat keras lebih kecil 16% dibanding dengan algoritme Grain v1 (Hamann, 2017).

Tabel 2.1. Daftar tinjauan pustaka.

Nama peneliti, tahun, judul	Metodologi	Hasil penelitian	Perbedaan
Sembiring, I., & Asang, M., 2017, Keamanan Data Pada Perangkat *Internet Of Things* Menggunakan Metode *Public-Key Cryptography*.	Penelitian ini mengimplementasikan algoritme RSA pada perangkat mikrokontroler Arduino Uno.	Algoritme RSA dapat berjalan dengan baik pada Arduino Uno meskipun membutuhkan waktu dan memori yang tidak sedikit.	Penelitian ini menggunakan kunci RSA sepanjang 16 bit. Kunci tersebut masih jauh dari panjang kunci RSA yang dianggap aman yaitu 1024 bit ke atas. Perangkat yang digunakan tidak memungkinkan untuk melakukan perhitungan dengan kunci sebesar itu.

Table 2.2. (Lanjutan) Daftar tinjauan pustaka.

Nama peneliti, tahun, judul	Metodologi	Hasil penelitian	Perbedaan
Laine, M., 2011, REST*ful Web Services for the Internet of Things*.	*Paper* ini menjelaskan tentang penerapan REST *web service* pada sistem pengiriman data antara dua perangkat IoT.	Arsitektur REST dapat diterapkan dengan baik pada sistem *machine to machine* serta berjalan secara efisien pada perangkat dengan daya yang terbatas.	Penelitian ini tidak menerapkan unsur keamanan *confidentiality* pada sistem.

Penelitian Implementasi Algoritme Kriptografi Lizard untuk Mengamankan Pengiriman Data Menggunakan Arsitektur Web Service REST pada Mikrokontroler NodeMCU berfokus pada pengimplementasian algoritme kriptografi pada perangkat mikrokontroler untuk memenuhi unsur kerahasiaan data pada sistem. Algoritme kriptografi *stream cipher* Lizard diharapkan mampu mengatasi masalah keterbatasan daya pada perangkat NodeMCU. Data pada perangkat NodeMCU akan dienkripsi menggunakan *keystream* dari algoritme Lizard terlebih dahulu sebelum dikirim menggunakan arsitektur komunikasi *web service* REST.

2.2 Dasar Teori

2.2.1 Kriptografi

Kriptografi merupakan suatu ilmu yang digunakan untuk mengamankan data, mengacak data sedemikian rupa hingga data tidak dapat dibaca oleh pihak yang tidak berwenang (Ekklesia, 2016). Terdapat dua proses utama pada kriptografi, yaitu enkripsi dan dekripsi. Proses enkripsi memiliki fungsi mengubah data asli (*plain text*) yang dapat terbaca dengan mudah menjadi bentuk *cipher text*. *Cipher text* merupakan suatu pesan yang telah dienkripsi dan tidak dapat terbaca dengan mudah. Sedangkan proses dekripsi sendiri merupakan kebalikan dari proses enkripsi. Proses enkripsi dan dekripsi dapat dilakukan dengan menggunakan kunci yang telah disepakati sebelumnya. Bentuk kunci tergantung pada algoritme kriptografi yang digunakan. Terdapat dua jenis kriptografi berdasarkan kuncinya, yaitu kriptografi kunci simetri dan asimetri.

Kriptografi kunci simetri menggunakan kunci yang identik pada proses enkripsi dan dekripsinya. Kunci ini bersifat sangat rahasia dan privat, hanya pengirim dan penerima saja yang memiliki kunci tersebut. Algoritme kriptografi jenis ini antara lain *DES* (*Data Encryption Standard*), Blowfish, Grain, Lizard dan lain sebagainya. Terdapat dua jenis kriptografi kunci simetri, yaitu *stream cipher* dan *block cipher*.

Stream cipher merupakan proses pengolahan keamanan data dengan mengenkripsi *plain text* menjadi *cipher text* menggunakan bit per bit atau setiap 1 bit pada setiap transformasi (1 karakter = 1 bita) (Nurrohmah, 2017). Algoritme kriptografi dengan tipe *stream cipher* memiliki dua nilai, yaitu berubah dan tidak berubah yang akan ditentukan oleh *keystream*. Algoritme dengan metode *stream cipher* antara lain Lizard, Grain, dan Trivium.

2.2.2 *Internet of Things* (IoT)

Internet of Things atau biasa disebut IoT adalah sebuah konsep di mana suatu objek mati memiliki kemampuan untuk menerima dan mengirimkan data melalui koneksi jaringan dengan keterlibatan pengguna yang minimal (Rose, 2015). Kata *things* dalam *internet of things* dapat didefinisikan sebagai subjek yang menghasilkan suatu data. Sejauh ini, IoT paling erat hubungannya dengan komunikasi *M2M* (*machine to machine*). Produk yang dibangun dengan kemampuan komunikasi *M2M* sering disebut dengan sistem cerdas atau biasa dikenal dengan sebutan *smart*, seperti *smart home*, *smart industries*, *smart energy, smart infrastructure*, dan sebagainya. Adapun tiga unsur utama pembentuk IoT yaitu:

2.2.2.1 *Constrained Device*

Constrained device merupakan komponen yang dikembangkan untuk tugas tertentu dan memiliki batasan terhadap daya dan komputasi (Rose, 2015). Salah satu contoh komponen ini adalah perangkat sensor dan mikrokontroler.

2.2.2.2 *Cloud Service*

Unsur ini berperan sebagai penyimpanan data dalam jumlah besar dan dapat terhubung dengan banyak *gateway* (Rose, 2015). *Cloud service* juga dapat melakukan pemantauan jarak jauh IoT-*driven* yang melibatkan pengumpulan data dari *things*, dan menggunakan data tersebut untuk memicu peringatan dan tindakan otomatis, seperti diagnosis jarak jauh, permintaan pemeliharaan, dan proses operasional lainnya.

2.2.2.3 *Gateway*

Gateway adalah sebuah komponen perantara antara sensor, perangkat, dan aplikasi yang menciptakan nilai dari data dan akses mereka (Vandikas, 2017). *Gateway* memungkinkan secara efisien pengumpulan dan pengamanan proses pengangkutan data dari perangkat, pengguna jarak jauh, dan aplikasi untuk melayani kebutuhan sistem.

2.2.3 Mikrokontroler NodeMCU 1.0 (ESP-12E *Module*)

NodeMCU 1.0 adalah salah satu dari banyak jenis mikrokontroler yang ada. NodeMCU 1.0 sendiri merupakan papan pengembangan mikrokontroler yang berbasis Arduino dengan menggunakan modul 802.11 bertipe ESP12-E yang telah terpasang di papan. NodeMCU 1.0 sangat kompatibel dengan bahasa pemrograman C dan komunikasi antarperangkat atau internet secara nirkabel.

NodeMCU memiliki dimensi sebesar 47 mm x 31 mm. Spesifikasi perangkat keras selebihnya dijelaskan pada Tabel 2.3. Tampilan fisik NodeMCU dapat dilihat pada Gambar 2.1.

Tabel 2.3. Spesifikasi perangkat NodeMCU 1.0.

Tegangan operasi	5 volt
Digital *I/O* (*Input*/*Output*) *pin*	30 buah
Arus *DC* (*Direct Current*) per *pin I/O*	20 mA
Memori *flash*	128 KB, 8 KB digunakan untuk *bootloader*.
***SRAM* (*Static Random Access Memory*)**	8 KB
***E^2PROM* (*Electrically Erasable Programmable Read-Only Memory*)**	4 KB
Storage	4 MB

Gambar 2.1. Tampilan fisik mikrokontroler NodeMCU 1.0.

2.2.4 Algoritme Lizard

Algoritme Lizard adalah sebuah algoritme kriptografi *stream cipher* yang tergolong ringan dan cocok digunakan pada perangkat yang memiliki keterbatasan daya dan memori (Hamann, 2017). Algoritme *stream cipher* ini memiliki kunci sebanyak 120 bit biner dan *IV* (*initial vector*) sebanyak 64 bit biner. Algoritme Lizard terbentuk dari 2 *NFSR* (*Non-linear Feedback Shift Register*) yang dinamakan NFSR1 dan NFSR2 dengan masing-masing berukuran 31 dan 90 *register*. Dari bit kunci dan *IV* yang dimasukkan ke dalam tiap *register* NFSR1 dan NFSR2 tersebut, terbentuklah *keystream*.

2.2.4.1 Komponen

Pada algoritme Lizard terdapat beberapa komponen penyusun utama untuk membentuk suatu *keystream*. Komponen utama tersebut adalah NFSR1, NFSR2, dan *output function* yang digambarkan dengan bentuk α.

NFSR1 merupakan sebuah *circular linked list* dengan 31 *register* yang berisi 1 bit biner tiap *node*-nya. *register* pada NFSR2 dilambangkan dalam S_i dengan i ∈ {0,....,30}. Tiap bit biner akan bergeser ke *register* yang bernomor lebih kecil setiap kali terjadi *clock*. Sedangkan, masukan pada *register* dengan nomor terbesar, digambarkan dengan f_1 pada Gambar 2.2 , memiliki rumus seperti berikut:

$$\begin{aligned} S_{30}^{t+1} = {} & \mathrm{S}_0^{\mathrm{t}} \oplus \mathrm{S}_2^{\mathrm{t}} \oplus \mathrm{S}_5^{\mathrm{t}} \oplus \mathrm{S}_6^{\mathrm{t}} \oplus \mathrm{S}_{15}^{\mathrm{t}} \oplus \mathrm{S}_{17}^{\mathrm{t}} \oplus \mathrm{S}_{18}^{\mathrm{t}} \oplus \mathrm{S}_{20}^{\mathrm{t}} \oplus \mathrm{S}_{25}^{\mathrm{t}} \oplus \mathrm{S}_8^{\mathrm{t}}\mathrm{S}_{18}^{\mathrm{t}} \\ & \oplus \mathrm{S}_8^{\mathrm{t}}\mathrm{S}_{20}^{\mathrm{t}} \oplus \mathrm{S}_{12}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}} \oplus \mathrm{S}_{14}^{\mathrm{t}}\mathrm{S}_{19}^{\mathrm{t}} \oplus \mathrm{S}_{17}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}} \oplus \mathrm{S}_{20}^{\mathrm{t}}\mathrm{S}_{22}^{\mathrm{t}} \oplus \mathrm{S}_4^{\mathrm{t}}\mathrm{S}_{12}^{\mathrm{t}}\mathrm{S}_{22}^{\mathrm{t}} \\ & \oplus \mathrm{S}_4^{\mathrm{t}}\mathrm{S}_{19}^{\mathrm{t}}\mathrm{S}_{22}^{\mathrm{t}} \oplus \mathrm{S}_7^{\mathrm{t}}\mathrm{S}_{20}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}} \oplus \mathrm{S}_8^{\mathrm{t}}\mathrm{S}_{18}^{\mathrm{t}}\mathrm{S}_{22}^{\mathrm{t}} \oplus \mathrm{S}_8^{\mathrm{t}}\mathrm{S}_{20}^{\mathrm{t}}\mathrm{S}_{22}^{\mathrm{t}} \oplus \mathrm{S}_{12}^{\mathrm{t}}\mathrm{S}_{19}^{\mathrm{t}}\mathrm{S}_{22}^{\mathrm{t}} \\ & \oplus \mathrm{S}_{20}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}}\mathrm{S}_{22}^{\mathrm{t}} \oplus \mathrm{S}_4^{\mathrm{t}}\mathrm{S}_7^{\mathrm{t}}\mathrm{S}_{12}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}} \oplus \mathrm{S}_4^{\mathrm{t}}\mathrm{S}_7^{\mathrm{t}}\mathrm{S}_{19}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}} \oplus \mathrm{S}_4^{\mathrm{t}}\mathrm{S}_{12}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}}\mathrm{S}_{22}^{\mathrm{t}} \\ & \oplus \mathrm{S}_4^{\mathrm{t}}\mathrm{S}_{19}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}}\mathrm{S}_{22}^{\mathrm{t}} \oplus \mathrm{S}_7^{\mathrm{t}}\mathrm{S}_8^{\mathrm{t}}\mathrm{S}_{18}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}} \oplus \mathrm{S}_7^{\mathrm{t}}\mathrm{S}_8^{\mathrm{t}}\mathrm{S}_{20}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}} \oplus \mathrm{S}_7^{\mathrm{t}}\mathrm{S}_{12}^{\mathrm{t}}\mathrm{S}_{19}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}} \\ & \oplus \mathrm{S}_8^{\mathrm{t}}\mathrm{S}_{18}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}}\mathrm{S}_{22}^{\mathrm{t}} \oplus \mathrm{S}_8^{\mathrm{t}}\mathrm{S}_{20}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}}\mathrm{S}_{22}^{\mathrm{t}} \oplus \mathrm{S}_{12}^{\mathrm{t}}\mathrm{S}_{19}^{\mathrm{t}}\mathrm{S}_{21}^{\mathrm{t}}\mathrm{S}_{22}^{\mathrm{t}} \end{aligned} \quad (2.1)$$

NFSR2 merupakan sebuah *circular linked list* dengan 90 *register* yang berisi 1 bit biner tiap nodenya. *register* pada NFSR2 dilambangkan dalam B_j dengan j ∈ {0,....,89}. Tiap bit biner akan bergeser ke *register* yang bernomor lebih kecil setiap kali terjadi *clock*. Sedangkan, masukan pada *register* dengan nomor terbesar, digambarkan dengan f_2 seperti pada Gambar 2.2, memiliki rumus seperti berikut:

$$\begin{aligned} B_{89}^{t+1} = {} & S_0^t \oplus \mathrm{B}_0^{\mathrm{t}} \oplus \mathrm{B}_{24}^{\mathrm{t}} \oplus \mathrm{B}_{49}^{\mathrm{t}} \oplus \mathrm{B}_{79}^{\mathrm{t}} \oplus \mathrm{B}_{84}^{\mathrm{t}} \oplus \mathrm{B}_3^{\mathrm{t}}\mathrm{B}_{59}^{\mathrm{t}} \oplus \mathrm{B}_{10}^{\mathrm{t}}\mathrm{B}_{12}^{\mathrm{t}} \\ & \oplus \mathrm{B}_{15}^{\mathrm{t}}\mathrm{B}_{16}^{\mathrm{t}} \oplus \mathrm{B}_{25}^{\mathrm{t}}\mathrm{B}_{53}^{\mathrm{t}} \oplus \mathrm{B}_{35}^{\mathrm{t}}\mathrm{B}_{42}^{\mathrm{t}} \oplus \mathrm{B}_{55}^{\mathrm{t}}\mathrm{B}_{58}^{\mathrm{t}} \oplus \mathrm{B}_{60}^{\mathrm{t}}\mathrm{B}_{74}^{\mathrm{t}} \\ & \oplus \mathrm{B}_{20}^{\mathrm{t}}\mathrm{B}_{22}^{\mathrm{t}}\mathrm{B}_{23}^{\mathrm{t}} \oplus \mathrm{B}_{62}^{\mathrm{t}}\mathrm{B}_{68}^{\mathrm{t}}\mathrm{B}_{72}^{\mathrm{t}} \oplus \mathrm{B}_{77}^{\mathrm{t}}\mathrm{B}_{80}^{\mathrm{t}}\mathrm{B}_{81}^{\mathrm{t}}\mathrm{B}_{83}^{\mathrm{t}} \end{aligned} \quad (2.2)$$

Output function, digambarkan dengan simbol α pada Gambar 2.2. *Output function* merupakan 1 bit biner keluaran yang akan membentuk 128 bit *keystream*. Menentukan keluaran dari α memiliki rumus seperti berikut:

$$Z_{\mathrm{t}} = L_{\mathrm{t}} \oplus \mathrm{Q}_{\mathrm{t}} \oplus \mathrm{T}_{\mathrm{t}} \oplus \check{\mathrm{T}}_{\mathrm{t}} \quad (2.3)$$

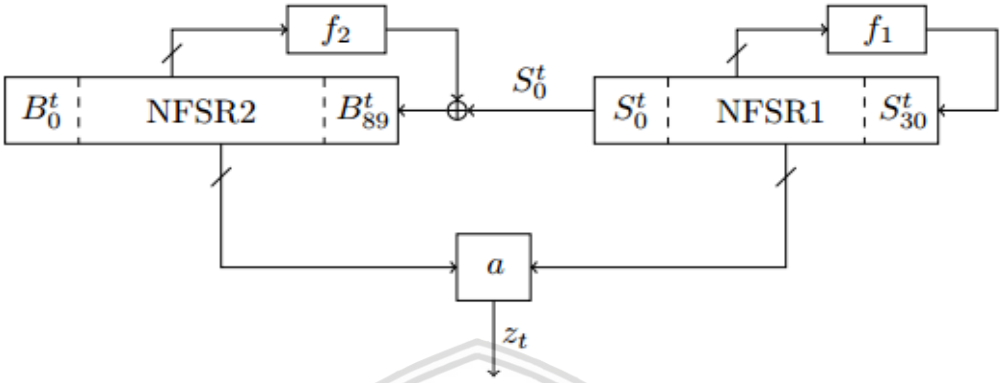
di mana:

$$L_{\mathrm{t}} = B_7^t \oplus B_{11}^t \oplus B_{30}^t \oplus B_{40}^t \oplus B_{45}^t \oplus B_{54}^t \oplus B_{71}^t \quad (2.4)$$

$$\mathrm{Q}_{\mathrm{t}} = \mathrm{B}_4^{\mathrm{t}}\mathrm{B}_{21}^{\mathrm{t}} \oplus \mathrm{B}_9^{\mathrm{t}}\mathrm{B}_{52}^{\mathrm{t}} \oplus \mathrm{B}_{18}^{\mathrm{t}}\mathrm{B}_{37}^{\mathrm{t}} \oplus \mathrm{B}_{44}^{\mathrm{t}}\mathrm{B}_{76}^{\mathrm{t}} \quad (2.5)$$

$$T_t = B_5^t \oplus B_8^t B_{82}^t \oplus B_{34}^t B_{67}^t B_{73}^t \oplus B_2^t B_{28}^t B_{41}^t B_{65}^t \oplus B_{13}^t B_{29}^t B_{50}^t B_{64}^t B_{75}^t \oplus B_6^t B_{14}^t B_{26}^t B_{32}^t B_{47}^t B_{61}^t \oplus B_1^t B_{19}^t B_{27}^t B_{43}^t B_{57}^t B_{66}^t B_{78}^t \quad (2.6)$$

$$\check{T}_t = S_{23}^t \oplus S_3^t S_{16}^t \oplus S_9^t S_{13}^t S_{48}^t \oplus S_1^t S_{24}^t S_{38}^t S_{63}^t \quad (2.7)$$



Gambar 2.2. Proses pembentukan *keystream* algoritme Lizard.
Sumber: Hamann, 2017.

2.2.4.2 Inisialisasi *state*

Proses inisialisasi *state* pada Lizard dibagi menjadi empat fase, yaitu fase *key and IV loading*, *grain-like mixing*, *second key addition*, dan *final diffusion*. Setiap fase menjalankan proses yang berbeda.

Fase *key and IV loading* merupakan fase pertama, yaitu dengan melakukan proses memuat atau memasukkan 120 bit kunci dan 64 bit *IV* ke dalam NFSR1 dan NFSR2. Rumus pembagiannya adalah sebagai berikut:

$$B_j = \begin{cases} K_j \oplus IV_j, \text{untuk } j \in \{0, \dots, 63\} \\ K_j, \quad \text{untuk } j \in \{64, \dots, 89\} \end{cases} \quad (2.8)$$

$$S_i = \begin{cases} K_{i+90}, & untuk\ i \in \{0, \dots, 28\} \\ K_{119} \oplus 1, & untuk\ i = 29 \\ 1, & untuk\ i = 30 \end{cases} \quad (2.9)$$

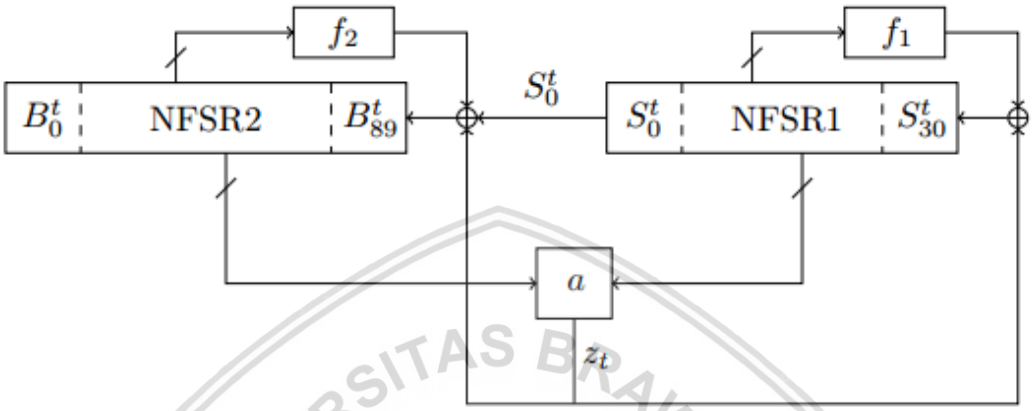
Pada fase ini, *cipher* akan di-*clock* sebanyak 128 kali tanpa membentuk *keystream* yang sesungguhnya. Pada Gambar 2.3, *keystream* atau Z_t akan dikembalikan lagi ke dalam *NFSR* sesuai dengan rumus yang ada. Rumus pergeseran dilambangkan dengan f_1 dan f_2 pada Gambar 2.3, untuk fase ini dengan $t \in \{0,\dots,127\}$ rumusnya adalah sebagai berikut:

$$B_j^{t+1} = B_{j+1}^t, \text{untuk } j \in \{0, \dots, 88\} \quad (2.10)$$

$$B_{89}^{t+1} = Z_t \oplus S_0^t \oplus B_0^t \oplus B_{24}^t \oplus B_{49}^t \oplus B_{79}^t \oplus B_{84}^t \oplus B_3^t B_{59}^t \oplus B_{10}^t B_{12}^t \oplus B_{15}^t B_{16}^t \oplus B_{25}^t B_{53}^t \oplus B_{35}^t B_{42}^t \oplus B_{55}^t B_{58}^t \oplus B_{60}^t B_{74}^t \oplus B_{20}^t B_{22}^t B_{23}^t \oplus B_{62}^t B_{68}^t B_{72}^t \oplus B_{77}^t B_{80}^t B_{81}^t B_{83}^t \quad (2.11)$$

$$S_i^{t+1} = S_{i+1}^t, \text{untuk } i \in \{0, \dots, 29\} \quad (2.12)$$

$$S_{30}^{t+1} = Z_t \oplus S_0^t \oplus S_2^t \oplus S_5^t \oplus S_6^t \oplus S_{15}^t \oplus S_{17}^t \oplus S_{18}^t \oplus S_{20}^t \oplus S_{25}^t \oplus S_8^t S_{18}^t \oplus S_8^t S_{20}^t \oplus S_{12}^t S_{21}^t \oplus S_{14}^t S_{19}^t \oplus S_{17}^t S_{21}^t \oplus S_{20}^t S_{22}^t \oplus S_4^t S_{12}^t S_{22}^t \oplus S_4^t S_{19}^t S_{22}^t \oplus S_7^t S_{20}^t S_{21}^t \oplus S_8^t S_{18}^t S_{22}^t \oplus S_8^t S_{20}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{22}^t \oplus S_{20}^t S_{21}^t S_{22}^t \oplus S_4^t S_7^t S_{12}^t S_{21}^t \oplus S_4^t S_7^t S_{19}^t S_{21}^t \oplus S_4^t S_{12}^t S_{21}^t S_{22}^t \oplus S_4^t S_{19}^t S_{21}^t S_{22}^t \oplus S_7^t S_8^t S_{18}^t S_{21}^t \oplus S_7^t S_8^t S_{20}^t S_{21}^t \oplus S_7^t S_{12}^t S_{19}^t S_{21}^t \oplus S_8^t S_{18}^t S_{21}^t S_{22}^t \oplus S_8^t S_{20}^t S_{21}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{21}^t S_{22}^t \quad (2.13)$$



Gambar 2.3. Proses inisialisasi *state* fase *grain-like mixing* algoritme Lizard. Sumber: Hamann, 2017.

Pada fase ini terjadi proses penambahan 120 bit kunci ke dalam NFSR1 dan NFSR2. 120 bit kunci tersebut merupakan kunci yang sama seperti pada fase 1. Penambahan kunci kedua dilakukan dengan rumus sebagai berikut:

$$B_j^{129} = B_j^{128} \oplus K_j, \text{untuk j} \in \{0, \dots, 89\} \quad (2.14)$$

$$S_i^{129} = \begin{cases} S_i^{128} \oplus K_{i+90}, \text{untuk i} \in \{0, \dots, 29\} \\ 1, \qquad \text{untuk i} = 30 \end{cases} \quad (2.15)$$

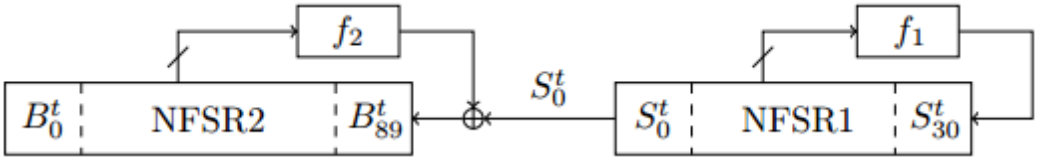
Fase *final diffusion* ini hampir sama dengan proses pada fase *grain-like mixing*, yaitu *cipher* akan di-*clock* sebanyak 128 kali. Perbedaan terletak pada proses *output function* dan pembentukan Z_t tidak dilakukan pada fase ini. Rumus pergeseran, dilambangkan dengan f_1 dan f_2 pada Gambar 2.4, untuk fase ini dengan t ∈ {129,....,256} rumusnya adalah sebagai berikut:

$$B_j^{t+1} = B_{j+1}^t, \text{untuk j} \in \{0, \dots, 88\} \quad (2.16)$$

$$B_{89}^{t+1} = S_0^t \oplus B_0^t \oplus B_{24}^t \oplus B_{49}^t \oplus B_{79}^t \oplus B_{84}^t \oplus B_3^t B_{59}^t \oplus B_{10}^t B_{12}^t \oplus B_{15}^t B_{16}^t \oplus B_{25}^t B_{53}^t \oplus B_{35}^t B_{42}^t \oplus B_{55}^t B_{58}^t \oplus B_{60}^t B_{74}^t \oplus B_{20}^t B_{22}^t B_{23}^t \oplus B_{62}^t B_{68}^t B_{72}^t \oplus B_{77}^t B_{80}^t B_{81}^t B_{83}^t \quad (2.17)$$

$$S_i^{t+1} = S_{i+1}^t, \text{untuk i} \in \{0, \dots, 29\} \quad (2.18)$$

$$S_{30}^{t+1} = S_0^t \oplus S_2^t \oplus S_5^t \oplus S_6^t \oplus S_{15}^t \oplus S_{17}^t \oplus S_{18}^t \oplus S_{20}^t \oplus S_{25}^t \oplus S_8^t S_{18}^t \oplus S_8^t S_{20}^t \oplus S_{12}^t S_{21}^t \oplus S_{14}^t S_{19}^t \oplus S_{17}^t S_{21}^t \oplus S_{20}^t S_{22}^t \oplus S_4^t S_{12}^t S_{22}^t \oplus S_4^t S_{19}^t S_{22}^t \oplus S_7^t S_{20}^t S_{21}^t \oplus S_8^t S_{18}^t S_{22}^t \oplus S_8^t S_{20}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{22}^t \oplus S_{20}^t S_{21}^t S_{22}^t \oplus S_4^t S_7^t S_{12}^t S_{21}^t \oplus S_4^t S_7^t S_{19}^t S_{21}^t \oplus S_4^t S_{12}^t S_{21}^t S_{22}^t \oplus S_4^t S_{19}^t S_{21}^t S_{22}^t \oplus S_7^t S_8^t S_{18}^t S_{21}^t \oplus S_7^t S_8^t S_{20}^t S_{21}^t \oplus S_7^t S_{12}^t S_{19}^t S_{21}^t \oplus S_8^t S_{18}^t S_{21}^t S_{22}^t \oplus S_8^t S_{20}^t S_{21}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{21}^t S_{22}^t \quad (2.19)$$



Gambar 2.4. Proses inisialisasi *state* fase *final diffusion* algoritme Lizard. Sumber: Hamann, 2017.

2.2.4.3 Pembentukan *Keystream*

Setelah proses inisialisasi *state* telah selesai, akan didapatkan *state* NFSR1 (S^{257}) dan NFSR2 (B^{257}) yang nantinya akan menjadi pembentuk *keystream* sesungguhnya. Pada Gambar 2.3, nilai Z_t atau bit *keystream* tidak perlu dikembalikan lagi ke dalam *NFSR*. Z_{257} merupakan bit pertama dari kumpulan bit *keystream*.

2.2.4.4 *Test Vector Keystream* Algoritme Lizard

Test vector merupakan pedoman untuk mengetahui suatu *keystream* yang dibentuk sesuai dengan *keystream* aslinya atau tidak. *Test vector* terbentuk dari kunci dan *IV* yang telah ditentukan sebelumnya. *Test vector* pada algoritme Lizard menurut Hamann (2017) adalah sebagai berikut:

1. Kunci: 0x00000000000000000000000000000000

   IV: 0x0000000000000000

   Keystream: 0xB6304CA4CA276B3355EC2E10968E84B3

2. Kunci: 0x00000000000000000FFFFFFFFFFFFFFF

   IV: 0xFFFFFFFFFFFFFFFF

   Keystream: 0x4D190941816F942358F0D164F4ECEB09

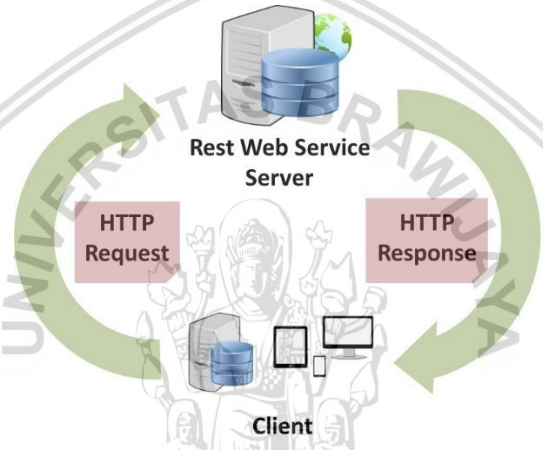
3. Kunci: 0x0123456789ABCDEF0123456789ABCD

   IV: 0xABCDEF0123456789

   Keystream: 0x983311A97831586548209DAFBF26FC93

2.2.5 *Web Service Representational State Transfer*

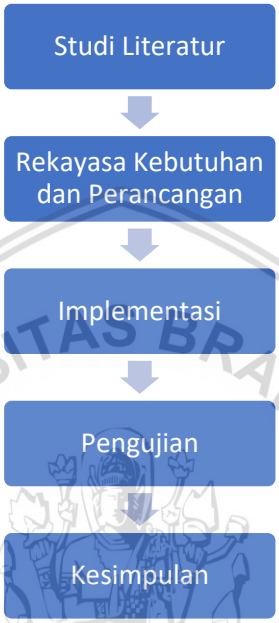
Web service merupakan suatu metode komunikasi *machine to machine* yang memanfaatkan protokol *Hypertext Transfer Protocol* (HTTP) (Richardson, 2007). Dapat dilihat pada Gambar 2.5, *web service* menggunakan mekanisme *request* dan *response* yang sama dengan *web page* yang diakses melalui *browser*. Terdapat beberapa jenis arsitektur pada *web service*, salah satunya adalah *Representational State Transfer* atau biasa disebut dengan REST. Pada arsitektur REST, server menyediakan data atau *resource* yang dapat berbentuk *Extensible Markup Language* (*XML*), *Javascript Object Notation* (JSON), ataupun data dengan tipe konvensional seperti *string* dan *integer* (Richardson, 2007). Klien REST dapat mengakses dan melakukan modifikasi pada *resource* tersebut melalui perintah HTTP *put*, *post*, *delete*, dan *get*. Setiap *resource* diidentifikasi oleh *Universal Resource Identifier* (*URI*).



Gambar 2.5. Mekanisme *request* dan *response resource* pada arsitektur REST. Sumber: Hafid Mukhlasin, 2015.

BAB 3 METODOLOGI PENELITIAN

Metodologi penelitian menjelaskan proses ilmiah apa saja yang diterapkan pada penelitian dengan tujuan untuk mendapatkan data yang berguna bagi hasil akhir penelitian ini. Dalam metodologi penelitian, proses dimulai dari tahap studi literatur hingga penarikan kesimpulan. Diagram alir metodologi penelitian ini dapat dilihat pada Gambar 3.1.



Gambar 3.1. Diagram alir metode penelitian.

3.1 Studi Literatur

Studi literatur digunakan untuk mempelajari berbagai referensi sebagai pengetahuan dasar dalam melakukan kegiatan perancangan, implementasi, dan pengujian dari penelitian ini. Referensi diperoleh dari artikel, buku, jurnal, serta penelitian-penelitian terkait, baik bertaraf nasional maupun internasional.

3.2 Rekayasa Kebutuhan dan Perancangan

Rekayasa kebutuhan dilakukan untuk mengatur perangkat keras dan perangkat lunak agar dapat mendukung berjalannya sistem yang akan dirancang dan diimplementasikan. Perancangan dilakukan untuk memberikan gambaran langkah kerja yang dilakukan dalam penelitian. Perancangan juga digunakan untuk mempermudah implementasi sesuai dengan tujuan penelitian. Teori-teori dari daftar pustaka digabungkan dengan ilmu yang diperoleh, dan selanjutnya diimplementasikan untuk merancang serta mengimplementasikan algoritme Lizard pada sistem IoT.

3.3 Implementasi

Implementasi dilakukan dengan merealisasikan hasil perancangan yang telah dibuat sebelumnya. Implementasi diawali dengan melakukan penyesuaian dan konfigurasi pada perangkat mikrokontroler NodeMCU, kemudian diteruskan dengan implementasi algoritme Lizard pada perangkat pembentuk sistem. Implementasi dilanjutkan dengan menambahkan berbagai fungsi yang diperlukan dalam proses enkripsi dan dekripsi data.

3.4 Pengujian

Pengujian dilakukan dengan empat tahap, yaitu pengujian validitas *keystream* algoritme kriptografi Lizard, pengujian fungsional, pengujian kinerja, dan pengujian keamanan. Variabel uji yang dimonitor adalah memori serta waktu yang diperlukan dalam menjalankan proses pembentukan *keystream* dan enkripsi data pada perangkat mikrokontroler NodeMCU. Pengujian validitas *keystream* dilakukan untuk mengetahui apakah *keystream* yang diperoleh telah sesuai dengan *test vector* algoritme Lizard yang dimiliki. Pengujian fungsional dilakukan untuk menguji setiap fungsi yang berjalan pada sistem. Pengujian kinerja dilakukan untuk mendapatkan hasil waktu dan penggunaan memori saat proses pembentukan *keystream* dan enkripsi data. Pengujian keamanan dilakukan untuk melihat hasil tangkapan paket pengiriman *cipher text*.

3.5 Kesimpulan

Penarikan kesimpulan dilakukan setelah semua tahapan mulai dari studi literatur, perancangan, implementasi, dan pengujian telah selesai dilakukan dan sesuai dengan kebutuhan yang ingin dicapai. Selanjutnya dapat diperoleh kesimpulan berdasarkan hasil penelitian yang telah dilakukan oleh penulis berupa implementasi algoritme kriptografi Lizard untuk mengamankan pengiriman data suhu dan kelembapan menggunakan arsitektur *web service* REST pada mikrokontroler NodeMCU. Kesimpulan diperoleh berdasarkan hasil pada tahap pengujian, sehingga dapat diperoleh penarikan kesimpulan untuk menjawab rumusan masalah yang telah disusun sebelumnya. Tahap terakhir dari penulisan adalah saran. Saran berguna untuk memberikan pertimbangan atas hasil yang telah dilakukan, serta untuk pengembangan dan pengimplementasian algoritme Lizard untuk keamanan pada sistem yang menggunakan perangkat dengan daya dan memori yang terbatas.

BAB 4 REKAYASA KEBUTUHAN

Bab rekayasa kebutuhan menjelaskan gambaran umum, serta daftar kebutuhan yang diperlukan untuk mendukung proses berjalannya sistem. Kebutuhan sistem tersebut meliputi kebutuhan fungsional dan nonfungsional perangkat lunak dan perangkat keras yang menyusun sistem.

4.1 Deskripsi Umum Sistem

Pada penelitian berjudul "implementasi algoritme kriptografi Lizard untuk mengamankan pengiriman data menggunakan arsitektur *web service* REST pada mikrokontroler NodeMCU" ini, terdapat dua mesin yang berperan sebagai klien REST dan server. Mesin yang berperan sebagai klien adalah NodeMCU, sedangkan server basis data sebagai server REST. Pada NodeMCU telah terpasang sensor DHT11 yang dapat mengukur suhu dan kelembapan. Klien secara periodik akan mengirim data suhu dan kelembapan tersebut kepada server.

Data suhu dan kelembapan dalam bentuk JSON akan dienkripsi terlebih dahulu dengan *keystream* dari algoritme Lizard yang dibentuk secara langsung pada perangkat NodeMCU. Hasil dari enkripsi tersebut adalah sebuah *cipher text*. *Cipher text* tersebut kemudian dikirim menuju server basis data yang akan melakukan dekripsi dengan *keystream* yang sama seperti pada sisi klien. Setelah proses dekripsi berhasil, data suhu dan kelembapan akan disimpan ke dalam basis data.

4.2 Kebutuhan Sistem

Penyusunan kebutuhan sistem ini bertujuan mengetahui kebutuhan yang diperlukan dalam membangun sistem agar dapat berjalan dengan baik dan benar. Kebutuhan sistem dapat memudahkan peneliti dalam melakukan proses perancangan dan implementasi. Terdapat dua jenis kebutuhan sistem di dalam penelitian ini, yaitu kebutuhan fungsional dan kebutuhan nonfungsional.

4.2.1 Kebutuhan Fungsional

Kebutuhan fungsional merupakan fitur-fitur yang harus dipenuhi dan dapat dilakukan oleh sistem. Kebutuhan fungsional sistem dalam penelitian ini dijelaskan pada Tabel 4.1.

Tabel 4.1. Tabel kebutuhan fungsional sistem.

No.	Kebutuhan Fungsional
1.	Perangkat NodeMCU dapat bertindak sebagai mikrokontroler bagi sensor DHT11 yang melakukan pembacaan suhu dan kelembapan.
2.	Perangkat NodeMCU dapat membentuk *keystream* dari algoritme Lizard.
3.	Perangkat NodeMCU dapat mengenkripsi JSON data suhu dan kelembapan menjadi sebuah *cipher text*.

Table 4.2. (Lanjutan) Tabel kebutuhan fungsional sistem.

No.	Kebutuhan Fungsional
4.	Perangkat NodeMCU dapat bertindak sebagai klien REST.
5.	Klien REST dapat mengirimkan *POST request* beserta *cipher text* kepada server.
6.	Perangkat NodeMCU dapat terhubung dengan jaringan internet secara nirkabel.
7.	Server basis data dapat bertindak sebagai server REST.
8.	Server REST dapat melakukan *handling POST request* dan menerima *cipher text* yang dikirimkan oleh klien.
9.	Server dapat mendekripsi *cipher text* yang diterima menjadi sebuah *plain text* yaitu JSON.
10.	Server dapat menyimpan data suhu dan kelembapan ke dalam basis data.
11.	Server basis data dapat terhubung dengan jaringan lokal secara nirkabel.

4.2.2 Kebutuhan Nonfungsional

Kebutuhan nonfungsional merupakan kebutuhan yang terfokus pada perilaku dan batasan fungsi pada sistem. Kebutuhan nonfungsional terdiri dari kebutuhan perangkat keras dan kebutuhan perangkat lunak.

4.2.2.1 Kebutuhan Perangkat Keras

Kebutuhan perangkat keras yang dibutuhkan untuk implementasi algoritme Lizard dan pengembangan sistem dalam penelitian ini dijelaskan pada Tabel 4.3.

Tabel 4.3. Tabel kebutuhan perangkat keras sistem.

Perangkat Keras	Keterangan
Sensor DHT11	Modul sensor untuk melakukan pembacaan suhu dan kelembapan.
NodeMCU 1.0 (ESP-12E *Module*)	Perangkat yang digunakan sebagai mikrokontroler bagi sensor DHT11. Selain itu, NodeMCU juga bertindak sebagai klien REST pada sistem.
Laptop ASUS	Perangkat laptop ini bertindak sebagai server basis data dan server REST.
Wi-Fi Router	Wi-Fi router untuk memberikan koneksi jaringan nirkabel pada NodeMCU dan ASUS A455L serta membentuk sebuah *network*.

4.2.2.2 Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak yang dibutuhkan untuk pengimplementasian algoritme Lizard dan pengembangan sistem dalam penelitian ini dijelaskan pada Tabel 4.4.

Tabel 4.4. Tabel kebutuhan perangkat lunak sistem.

Perangkat Lunak	Keterangan
Arduino IDE	Aplikasi untuk tempat menuliskan baris-baris kode pemrograman ".ino" yang dapat diunggah ke perangkat NodeMCU.
MySQL-server	Aplikasi basis data yang digunakan untuk menyimpan data suhu dan kelembapan.
Sublime Text 3	Aplikasi untuk tempat menuliskan baris-baris kode pemrograman Python yang akan membentuk server REST.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

5.1 Perancangan Sistem

Perancangan digunakan sebagai acuan saat melakukan proses implementasi sistem sehingga dapat memudahkan implementasi berjalan secara terstruktur sesuai dengan yang diinginkan. Perancangan sistem terdiri dari perancangan alur kerja sistem, perancangan REST *web service* klien, perancangan REST *web service* server, perancangan skema basis data, dan perancangan mekanisme pada saat proses pengujian sistem.

5.1.1 Perancangan Algoritme Kriptografi Lizard

Algoritme kriptografi Lizard memiliki dua proses utama, yaitu proses inisialisasi *state* dan proses pembentukan *keystream* sebenarnya. Proses inisialisasi *state* terdiri dari empat fase, yaitu fase memuat kunci dan *IV* ke dalam *resgister* NFSR1 dan NFSR2, fase *grain-like mixing*, fase *second key add*, dan fase *final diffusion*. Setelah proses inisialisasi *state* selesai, dimulailah proses pembentukan *keystream* algoritme Lizard yang sebenarnya. Contoh manualisasi hasil perhitungan di tiap proses hingga terbentuknya *keystream* dijabarkan pada Tabel 5.1 di bawah.

Tabel 5.1. Manualisasi pembentukan *keystream* algoritme Lizard.

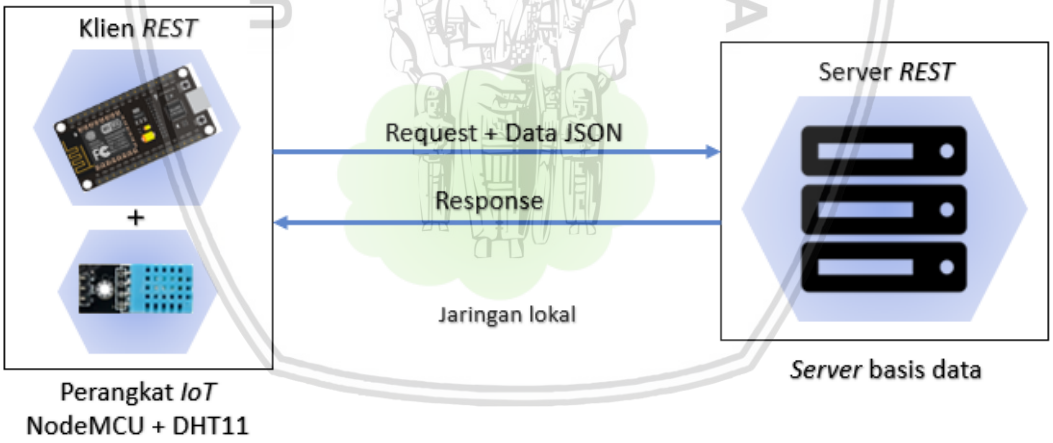
Fase	Hasil manualisasi dalam bentuk biner
Deklarasi kunci dan *IV*	**Kunci**: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 (120 bit) ***IV***: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 (64 bit)
Pemuatan kunci dan *IV* ke dalam *register*	**NFSR1**: 00000000 00000000 00000000 0000011 (31 bit) **NFSR2**: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00 (90 bit)
Grain-like mixing	**NFSR1**: 11110000 10100010 01101100 0001101 (31 bit) **NFSR2**: 11001010 00011010 00101010 01001010 10010110 01001101 00001001 01110100 01101111 11100101 11000000 00000001
Second key add	**NFSR1**: 11110000 10100010 01101100 0001101 (31 bit) **NFSR2**: 11001010 00011010 00101010 01001010 10010110 01001101 00001001 01110100 01101111 11100101 11000000 01 (90 bit)

Table 5.2. (Lanjutan) Manualisasi pembentukan *keystream* algoritme Lizard.

Fase	Hasil manualisasi dalam bentuk biner
Final diffusion	**NFSR1**: 10110101 10010110 11110100 1001001 (31 bit) **NFSR2**: 01111110 11110000 01001011 01101011 11101000 00000101 00111111 01100111 11011101 00111101 01100111 01 (90 bit)
Pembentukan *keystream* sebenarnya	***Keystream***: 10110110 00110000 01001100 10100100 11001010 00100111 01101011 00110011 01010101 11101100 00101110 00010000 10010110 10001110 10000100 10110011 (128 bit)

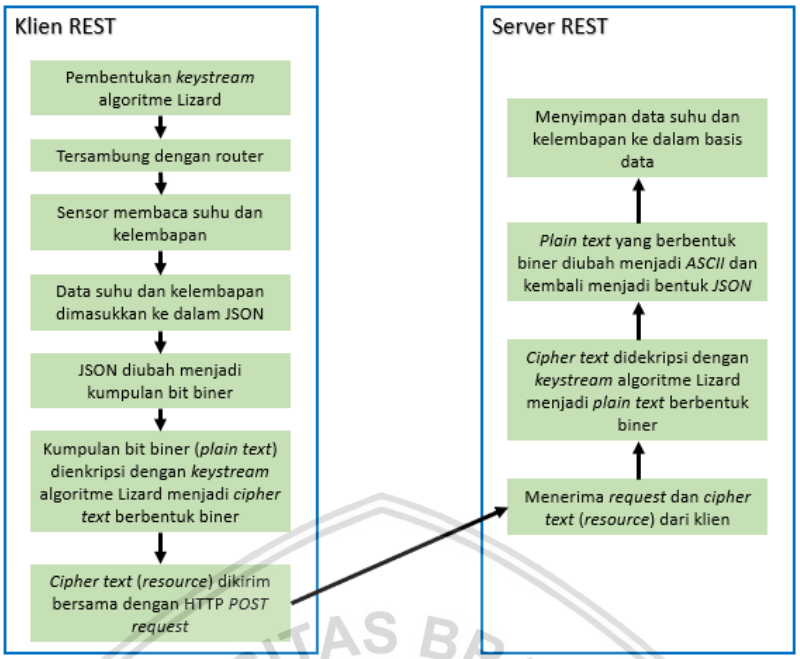
5.1.2 Perancangan Alur Kerja Sistem

Gambar 5.1 menjelaskan tentang rancangan proses komunikasi antarperangkat yang ada pada sistem. Secara umum terlihat bahwa sistem menerapkan arsitektur komunikasi *web service* REST. Terdapat dua unsur pembentuk dalam arsitektur *web service* yaitu klien dan server. Sistem akan menggunakan jaringan lokal dalam proses komunikasi antara klien dengan server. Pada sistem ini, klien REST akan mengirimkan *request POST* beserta data JSON kepada server REST. Server bertugas untuk menerima *request* dan data dari klien tersebut. Data lalu diproses sesuai dengan *URL* yang diakses oleh klien.



Gambar 5.1. Rancangan proses komunikasi klien-server REST.

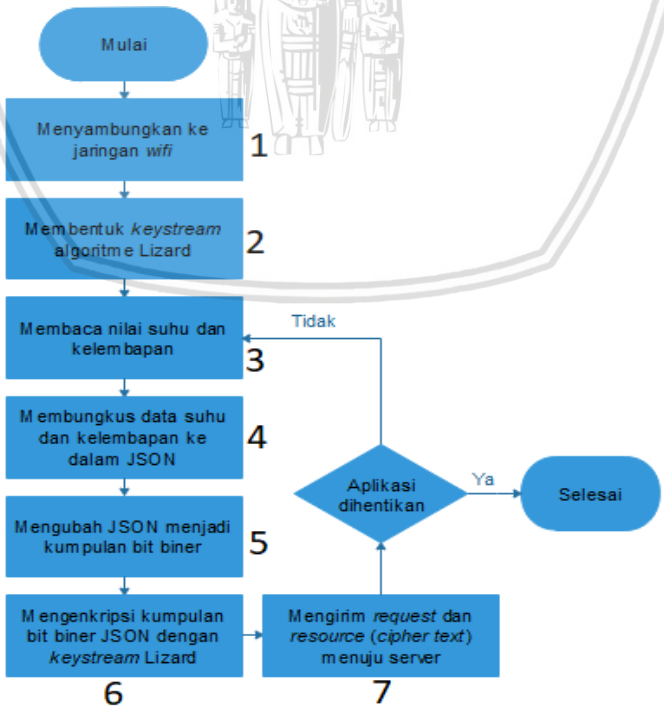
Gambar 5.2 menjelaskan tentang rancangan alur berjalannya data dan kerja sistem. Sistem dirancang untuk dapat membentuk *keystream* algoritme Lizard, tersambung ke router, serta memproses data suhu dan kelembapan dari sensor DHT11 hingga tersimpan di basis data. Secara garis besar, proses akan dimulai dari pembacaan nilai suhu dan kelembapan, lalu data tersebut akan dienkripsi terlebih dahulu sebelum dikirim menuju ke server. Pada sisi server, data yang diterima akan didekripsi terlebih dahulu dan disimpan ke dalam basis data apabila proses dekripsi telah berhasil.



Gambar 5.1. Rancangan alur kerja sistem.

5.1.3 Perancangan REST *Web Service* Klien

Pada rancangan sistem ini, mikrokontroler NodeMCU akan bertindak sebagai klien REST. Mikrokontroler dirancang untuk dapat membentuk *keystream* algoritme Lizard, tersambung dengan jaringan nirkabel, serta memproses data suhu dan kelembapan hingga siap dikirim dalam bentuk *cipher text* menuju server.



Gambar 5.2. Rancangan alur kerja klien REST.

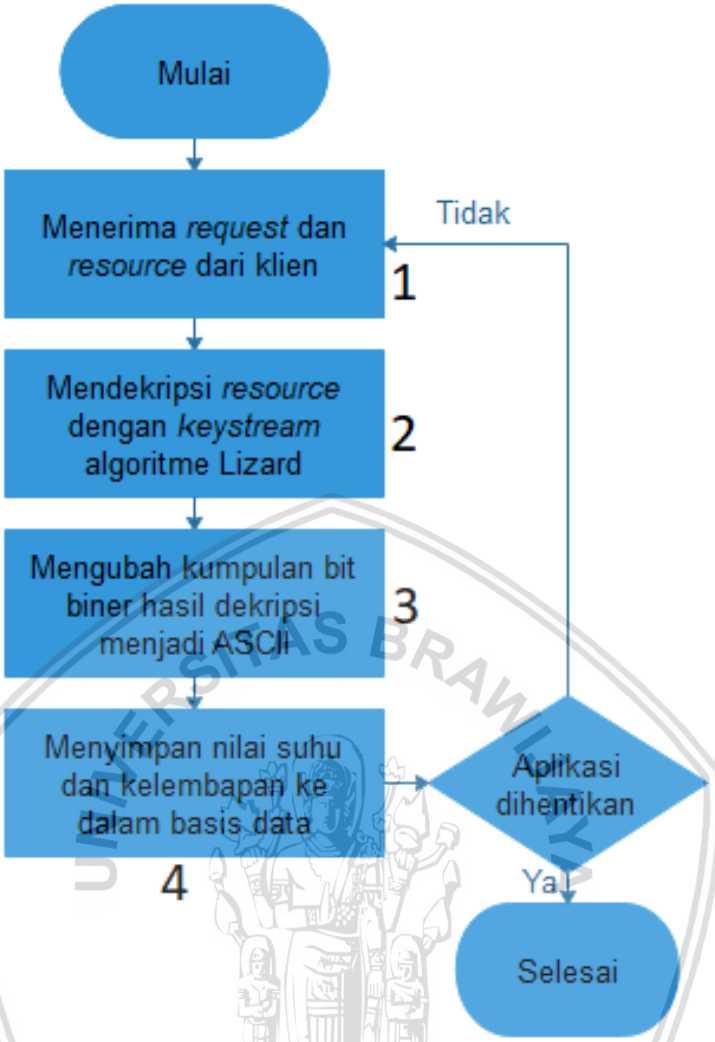
Keterangan dari tiap proses pada diagram alir pada Gambar 5.3 di atas terdapat pada Tabel 5.3 di bawah ini.

Tabel 5.3. Penjelasan rancangan proses pada diagram alir klien REST.

No	Fungsi	Keterangan
1	Menyambungkan ke jaringan wifi	Mikrokontroler terhubung dengan jaringan nirkabel melalui *router*.
2	Membentuk *keystream* algoritme Lizard	Membentuk *keystream* algoritme Lizard berdasarkan variasi kunci dan *IV* yang diingkan. Pembentukan *keystream* hanya dilakukan satu kali saat sistem mulai berjalan.
3	Membaca nilai suhu dan kelembapan	Mikrokontroler membaca nilai suhu dan kelembapan yang diperoleh dari sensor DHT11. Nilai suhu dan kelembapan tersebut disimpan pada masing-masing variabel.
4	Membungkus data suhu dan kelembapan ke dalam JSON	Membungkus variabel suhu dan kelembapan beserta nilainya ke dalam JSON.
5	Mengubah JSON menjadi bentuk biner	Mengubah tiap karakter ASCII dalam JSON ke dalam bentuk biner. Hasil biner dari tiap karakter lalu digabung menjadi sebuah kumpulan bit biner.
6	Mengenkripsi *plain text* menjadi *cipher text*	Mengenkripsi kumpulan bit biner (*plain text*) dengan *keystream* algoritme Lizard. Proses enkripsi menghasilkan sebuah *cipher text* yang berbentuk kumpulan bit biner.
7	Mengirim HTTP *request* dan *cipher text menuju* server	Mengirim *request* dan *resource* menuju server. *Resource* pada arsitektur *web service* REST merupakan data yang dipertukarkan antara klien dengan server. Dalam sistem ini *cipher text* adalah *resource*. Setelah itu, sistem akan kembali ke proses nomor tiga setelah 30 detik.

5.1.4 Perancangan REST *Web Service* Server

Pada rancangan sistem ini, server basis data akan bertindak sebagai server REST. Server REST akan ditulis dalam bahasa pemrograman Python dan dirancang untuk dapat menerima HTTP *request* dari klien serta memproses data yang diterima hingga tersimpan di basis data.



Gambar 5.3. Rancangan alur kerja server REST.

Keterangan dari tiap proses pada diagram alir pada Gambar 5.4 di atas terdapat pada Tabel 5.4 di bawah ini.

Tabel 5.4. Penjelasan rancangan proses pada diagram alir server REST.

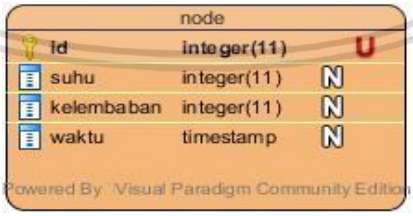
No	Fungsi	Keterangan
1	Menerima HTTP *request* dan cipher text dari klien	Jika ada *request* yang diterima dari klien, server akan melakukan penanganan sesuai dengan *URL* dari *request* yang diterima tersebut.

Table 5.5. (Lanjutan) Penjelasan rancangan proses pada diagram alir server REST.

No	Fungsi	Keterangan
2	Mendekripsi *cipher text menjadi plain text*	Melakukan dekripsi pada *resource* yang diterima dengan *keystream* algoritme Lizard yang sama seperti pada sisi klien. *Resource* pada sistem ini adalah *cipher text* berbentuk kumpulan bit biner.
3	Mengubah kumpulan bit biner menjadi bentuk ASCII	Mengubah bentuk bit biner dari hasil dekripsi menjadi karakter ASCII. Kumpulan karakter ASCII tersebut membentuk sebuah JSON yang sama seperti pada sisi klien sebelum data tersebut diubah menjadi bentuk biner.
4	Menyimpan data ke dalam basis data	Menyimpan data suhu dan kelembapan yang terdapat dalam JSON ke dalam basis data yang telah disiapkan sebelumnya.

5.1.5 Perancangan Skema Basis data

Pada bagian perancangan skema basis data, data suhu dan kelembapan yang telah berhasil didekripsi akan disimpan pada basis data MySQL-Server. Nilai suhu dan kelembapan yang bertipe *integer* dari data tersebut akan disimpan dalam kolom "suhu" dan "kelembapan". Gambar 5.5 menunjukkan entitas basis data pada sistem ini. Terdapat satu tabel yang terdiri dari empat kolom, yaitu "id", "suhu", "kelembapan", dan "waktu". Kolom id akan dibentuk secara *auto-increment* dan menyimpan nomor identitas dari data suhu dan kelembapan. Kolom suhu dan kelembapan menyimpan data bertipe *integer*. Kolom waktu yang bertipe *timestamp* akan menyimpan waktu saat data diterima oleh server dengan format "tahun-bulan-tanggal jam:menit:detik".



Gambar 4.5. Rancangan entitas basis data.

5.1.6 Perancangan Pengujian

Pengujian pada sistem akan dibagi dalam empat bagian yaitu pengujian validasi *keystream* algoritme Lizard, fungsional, kinerja, dan keamanan. Pengujian validasi *keystream* dilakukan untuk memastikan bahwa *keystream* yang telah dibentuk pada sistem sesuai dengan *test vector*. Penjelasan dari pengujian *keystream* dapat dilihat pada Tabel 5.6 di bawah.

Tabel 5.6. Penjelasan perancangan pengujian validasi *keystream*.

No	Kode	Nama uji	Skenario
1	PK-01	Pengujian validasi *keystream*	1. Membandingkan hasil pembentukan *keystream* algoritme Lizard pada perangkat NodeMCU dengan *test vector*.

Pengujian fungsional akan menguji fungsi-fungsi yang ada dalam sistem apakah sudah berjalan dengan benar atau tidak. Penjelasan dari pengujian fungsional dapat dilihat pada Tabel 5.7 di bawah.

Tabel 5.7. Penjelasan perancangan pengujian fungsional sistem.

No	Kode	Fungsi	Skenario
1	PF-01	Menyambungkan ke jaringan wifi	1. Terdapat jaringan *wifi* yang aktif di sekitar perangkat NodeMCU. 2. Memasukkan nama dan *password* jaringan *wifi*. 3. Menampilkan alamat *IP* (*Internet Protocol*) yang diperoleh.
2	PF-02	Membentuk *keystream* algoritme Lizard	1. Perangkat NodeMCU tersambung dengan jaringan *wifi*. 2. Melihat hasil pembentukan *keystream*.
3	PF-03	Membaca nilai suhu dan kelembapan	1. Perangkat NodeMCU tersambung dengan jaringan *wifi*. 2. Sensor DHT11 tersambung dengan mikrokontroler NodeMCU. 3. Menampilkan nilai suhu dan kelembapan.
4	PF-04	Membungkus data suhu dan kelembapan ke dalam JSON	1. Perangkat NodeMCU tersambung dengan jaringan *wifi*. 2. Sensor DHT11 tersambung dengan mikrokontroler NodeMCU. 3. Menampilkan data JSON.

Table 5.8. (Lanjutan) Penjelasan perancangan pengujian fungsional sistem.

No	Kode	Fungsi	Skenario
5	PF-05	Mengubah JSON menjadi bentuk biner	1. Perangkat NodeMCU tersambung dengan jaringan *wifi*. 2. Sensor DHT11 tersambung dengan mikrokontroler NodeMCU. 3. Menampilkan hasil pengubahan JSON menjadi bentuk biner.
6	PF-06	Mengenkripsi *plain text* menjadi *cipher text*	1. Perangkat NodeMCU tersambung dengan jaringan *wifi*. 2. Sensor DHT11 tersambung dengan mikrokontroler NodeMCU. 3. *Keystream* berhasil dibentuk. 4. Berhasil mengubah JSON menjadi bentuk biner. 5. Menampilkan hasil enkripsi.
7	PF-07	Mengirim HTTP *request* dan *cipher text menuju* server	1. Perangkat NodeMCU tersambung dengan jaringan *wifi*. 2. Sensor DHT11 tersambung dengan mikrokontroler NodeMCU. 3. Berhasil mengenkripsi *plain text*menjadi *cipher text*. 4. Memasukkan alamat IP dan *port* server REST beserta *URL*-nya. 5. Melihat hasil pengiriman pada server REST tujuan.
8	PF-08	Menerima HTTP *request* dan cipher text dari klien	1. Aplikasi basis data aktif. 2. Server REST aktif. 3. Menampilkan hasil penerimaan.
9	PF-09	Mendekripsi *cipher text* menjadi *plain text*	1. Aplikasi basis data aktif. 2. Server REST aktif. 3. Berhasil menerima HTTP *request* dan *cipher text*. 4. Menampilkan hasil dekripsi.

Table 5.9. (Lanjutan) Penjelasan perancangan pengujian fungsional sistem.

No	Kode	Fungsi	Skenario
10	PF-10	Mengubah kumpulan bit biner menjadi bentuk ASCII	1. Aplikasi basis data aktif. 2. Server REST aktif. 3. Berhasil menerima HTTP *request* dan *cipher text*. 4. Berhasil melakukan proses dekripsi. 5. Menampilkan hasil pengubahan biner menjadi bentuk ASCII.
11	PF-11	Menyimpan data ke dalam basis data	1. Aplikasi basis data aktif. 2. Server REST aktif. 3. Berhasil menerima HTTP *request* dan *cipher text*. 4. Berhasil mengubah biner menjadi bentuk ASCII. 5. Melihat data pada basis data.

Pengujian kinerja dibagi menjadi dua bagian yaitu pengujian waktu dan memori pada proses pembentukan *keystream* dan pengujian waktu dan memori pada proses enkripsi. Penjelasan dari pengujian kinerja dapat dilihat pada Tabel 5.10 di bawah.

Tabel 5.10. Penjelasan perancangan pengujian kinerja perangkat NodeMCU.

No	Kode	Nama uji	Skenario
1	PK-01	Pengujian waktu dan memori pada proses pembentukan *keystream*	1. Melihat waktu yang diperlukan oleh perangkat NodeMCU untuk membentuk *keystream* dengan menggunakan fungsi micros(). 2. Melihat memori yang diperlukan oleh perangkat NodeMCU untuk membentuk *keystream* dengan menggunakan fungsi ESP.getFreeHeap(). 3. Pengujian pada skenario ini akan dilakukan sebanyak 50 kali proses dan hasilnya akan dirata-rata.

Table 5.11. (Lanjutan) Penjelasan perancangan pengujian kinerja perangkat NodeMCU.

No	Kode	Nama uji	Skenario
2	PK-02	Pengujian waktu dan memori pada proses enkripsi	1. Melihat waktu yang diperlukan oleh perangkat NodeMCU untuk melakukan enkripsi *plain text* dengan menggunakan fungsi micros(). 2. Melihat memori yang diperlukan oleh perangkat NodeMCU untuk melakukan enkripsi *plain text* dengan menggunakan fungsi ESP.getFreeHeap(). 3. Pengujian pada skenario ini akan dilakukan sebanyak 50 kali proses dan hasilnya akan dirata-rata.

Pengujian keamanan data dilakukan untuk memastikan bahwa sistem telah memenuhi unsur kerahasiaan data atau *confidentiality*. Penjelasan dari pengujian *keystream* dapat dilihat pada Tabel 5.12 di bawah.

Tabel 5.12. Penjelasan perancangan pengujian kerahasiaan data.

No	Kode	Nama uji	Skenario
1	PS-01	Pengujian kerahasiaan data saat proses pengiriman	1. Melihat hasil tangkapan paket pada proses pengiriman data dari klien REST menuju server menggunakan aplikasi Wireshark. 2. Membandingkan penangkapan paket saat sistem mengirim *plain text* dan saat sistem mengirim *cipher text*.

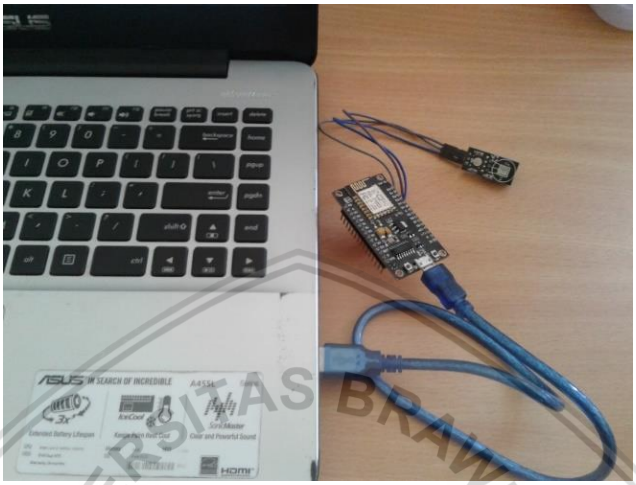
5.2 Implementasi Sistem

Pada subbab implementasi sistem dijelaskan tentang pengimplementasian dari setiap bagian yang akan membentuk sistem secara utuh. Proses implementasi dilakukan sesuai dengan perancangan yang telah dijelaskan pada subbab perancangan sistem.

5.2.1 Implementasi Perangkat Keras

Terdapat tiga komponen utama perangkat keras yang membentuk sistem ini. Perangkat keras tersebut yaitu sensor DHT11, mikrokontroler NodeMCU, dan laptop yang bertindak sebagai server REST. Sensor DHT11 terhubung menggunakan kabel *jumper* dengan perangkat NodeMCU melalui pin-pin yang ada

pada kedua perangkat tersebut. Pin positif pada DHT11 terhubung dengan pin 3V pada NodeMCU. Pin negatif pada DHT11 terhubung dengan pin GND pada NodeMCU. Pin “out” pada DHT11 terhubung dengan pin D4 pada NodeMCU. Perangkat mikrokontroler NodeMCU terhubung dengan laptop seperti pada Gambar 5.6 agar mendapat sumber daya listrik dan dapat dilakukan proses unggah kode program pada mikrokontroler.



Gambar 5.5. Implementasi perangkat keras.

5.2.2 Implementasi Perangkat Lunak

Arduino IDE merupakan perangkat lunak yang digunakan untuk menulis kode program yang dijalankan pada perangkat mikrokontroler NodeMCU. Perlu beberapa konfigurasi pada pengaturan *board* di Arduino IDE agar pengguna dapat mengunggah kode program ke perangkat mikrokontroler dengan benar. Pengaturan tersebut dapat dilihat pada Tabel 5.13 berikut:

Tabel 5.13. Pengaturan perangkat lunak Arduino IDE.

Board	NodeMCU 1.0 (ESP12-E Module)
Flash size	4M (3M SPIFFS)
Debug port	Serial
Debug level	None
IwIP variant	v2 Prebuilt (MSS=536)
CPU frequency	80 MHz
Upload speed	115200

Kode program *web service* server REST ditulis dengan bahasa pemrograman Python. Kode program menggunakan *library* Flask untuk menjalankan *web service* dan *library* mysql.connector untuk koneksi dengan basis data MySQL-Server. Sistem ini menggunakan Python versi 2.7.13.

5.2.3 Implementasi Algoritme Kriptografi Lizard

Algoritme kriptografi Lizard diimplementasikan pada perangkat mikrokontroler NodeMCU. Kode program algoritme Lizard ditulis dengan konsep pemrograman berorientasi objek di mana terdapat atribut dan fungsi di dalamnya. Hasil implementasi algoritme kriptografi Lizard dapat dilihat pada Tabel 5.14.

Tabel 5.14. *Pseudocode* implementasi algoritme Lizard.

```
1   fase key and IV loading
2   for j = 0 to j <= 63 do
3      nfsr2[j] ← key[j] ⊕ iv[j]
4   end for
5   for j = 64 to j <= 89 do
6      nfsr2[j] ← key[j]
7   end for
8   for i = 0 to i <= 28 do
9      nfsr1[i] ← key[i+90]
10  end for
11  nfsr1[29] ← key[i+90] ⊕ 1
12  nfsr1[30] ← 1
13
14  fase grain-like-mixing
15  for t = 0 to t <= 127 do
16     nfsr1[30] = keystream ⊕ nfsr1[0] ⊕ nfsr1[5] ⊕ nfsr1[6] ⊕
17  nfsr1[15] ⊕ nfsr1[17] ⊕ nfsr1[18] ⊕ nfsr1[20] ⊕ nfsr1[25] ⊕
18  nfsr1[8]&nfsr1[18] ⊕ nfsr1[8]&nfsr1[20] ⊕ nfsr1[12]&nfsr1[21] ⊕
19  nfsr1[14]&nfsr1[19] ⊕ nfsr1[17]&nfsr1[21] ⊕ nfsr1[20]&nfsr1[22]
20  ⊕ nfsr1[4]&nfsr1[12]&nfsr1[22] ⊕ nfsr1[4]&nfsr1[19]&nfsr1[22] ⊕
21  nfsr1[7]&nfsr1[20]&nfsr1[21] ⊕ nfsr1[8]&nfsr1[18]&nfsr1[22] ⊕
22  nfsr1[8]&nfsr1[20]&nfsr1[22] ⊕ nfsr1[12]&nfsr1[19]&nfsr1[22] ⊕
23  nfsr1[20]&nfsr1[21]&nfsr1[22]⊕
24  nfsr1[4]&nfsr1[7]&nfsr1[12]&nfsr1[21] ⊕
25  nfsr1[4]&nfsr1[7]&nfsr1[19]&nfsr1[21]⊕
26  nfsr1[4]&nfsr1[12]&nfsr1[21]&nfsr1[22] ⊕
27  nfsr1[4]&nfsr1[19]&nfsr1[21]&nfsr1[22] ⊕
28  nfsr1[7]&nfsr1[8]&nfsr1[18]&nfsr1[21] ⊕
29  nfsr1[7]&nfsr1[8]&nfsr1[20]&nfsr1[21] ⊕
30  nfsr1[7]&nfsr1[12]&nfsr1[19]&nfsr1[21] ⊕
31  nfsr1[8]&nfsr1[18]&nfsr1[21]&nfsr1[22] ⊕
32  nfsr1[8]&nfsr1[20]&nfsr1[21]&nfsr1[22] ⊕
33  nfsr1[12]&nfsr1[19]&nfsr1[21]&nfsr1[22]
34     for i = 0 to i <= 29 do
35        nfsr1[i] = nfsr1[i+1]
36     end for
37     nfsr2[89] = keystream ⊕ nfsr1[0] ⊕ nfsr2[0] ⊕ nfsr2[24] ⊕
38  nfsr2[49] ⊕ nfsr2[79] ⊕ nfsr2[84] ⊕ nfsr2[3]&nfsr2[59] ⊕
39  nfsr2[10]&nfsr2[12] ⊕ nfsr2[15]&nfsr2[16] ⊕ nfsr2[25]&nfsr2[53]
40  ⊕ nfsr2[35]&nfsr2[42] ⊕ nfsr2[55]&nfsr2[58] ⊕
41  nfsr2[60]&nfsr2[74] ⊕ nfsr2[20]&nfsr2[22]&nfsr2[23] ⊕
42  nfsr2[62]&nfsr2[68]&nfsr2[72] ⊕
43  nfsr2[77]&nfsr2[80]&nfsr2[81]&nfsr2[83]
44     for j = 0 to j <= 88 do
45        nfsr2[j] = nfsr2[j+1]
46     end for
47  end for
48
```

Tabel 5.15. (Lanjutan) *Pseudocode* implementasi algoritme Lizard.

```
49  fase second key add
50  for t = 128 do
51     for j = 0 to j <= 89 do
52        nfsr2[j] = nfsr2[j] ⊕ key[j]
53     end for
54     for i = 0 to i <= 29 do
55        nfsr1[i] = nfsr1[i] ⊕ key[i+90]
56     end for
57     nfsr1[30]=1
58  end for
59
60  fase final diffusion
61  for t = 129 to t <= 256 do
62     nfsr1[30] = nfsr1[0] ⊕ nfsr1[5] ⊕ nfsr1[6] ⊕ nfsr1[15] ⊕
63  nfsr1[17] ⊕ nfsr1[18] ⊕ nfsr1[20] ⊕ nfsr1[25] ⊕
64  nfsr1[8]&nfsr1[18] ⊕ nfsr1[8]&nfsr1[20] ⊕ nfsr1[12]&nfsr1[21] ⊕
65  nfsr1[14]&nfsr1[19] ⊕ nfsr1[17]&nfsr1[21] ⊕ nfsr1[20]&nfsr1[22]
66  ⊕ nfsr1[4]&nfsr1[12]&nfsr1[22] ⊕ nfsr1[4]&nfsr1[19]&nfsr1[22] ⊕
67  nfsr1[7]&nfsr1[20]&nfsr1[21] ⊕ nfsr1[8]&nfsr1[18]&nfsr1[22] ⊕
68  nfsr1[8]&nfsr1[20]&nfsr1[22] ⊕ nfsr1[12]&nfsr1[19]&nfsr1[22] ⊕
69  nfsr1[20]&nfsr1[21]&nfsr1[22] ⊕
70  nfsr1[4]&nfsr1[7]&nfsr1[12]&nfsr1[21] ⊕
71  nfsr1[4]&nfsr1[7]&nfsr1[19]&nfsr1[21]⊕
72  nfsr1[4]&nfsr1[12]&nfsr1[21]&nfsr1[22] ⊕
73  nfsr1[4]&nfsr1[19]&nfsr1[21]&nfsr1[22] ⊕
74  nfsr1[7]&nfsr1[8]&nfsr1[18]&nfsr1[21] ⊕
75  nfsr1[7]&nfsr1[8]&nfsr1[20]&nfsr1[21] ⊕
76  nfsr1[7]&nfsr1[12]&nfsr1[19]&nfsr1[21] ⊕
77  nfsr1[8]&nfsr1[18]&nfsr1[21]&nfsr1[22] ⊕
78  nfsr1[8]&nfsr1[20]&nfsr1[21]&nfsr1[22] ⊕
79  nfsr1[12]&nfsr1[19]&nfsr1[21]&nfsr1[22]
80     for i = 0 to i <= 29 do
81        nfsr1[i] = nfsr1[i+1]
82     end for
83     nfsr2[89] = nfsr1[0] ⊕ nfsr2[0] ⊕ nfsr2[24] ⊕ nfsr2[49] ⊕
84  nfsr2[79] ⊕ nfsr2[84] ⊕ nfsr2[3]&nfsr2[59] ⊕ nfsr2[10]&nfsr2[12]
85  ⊕ nfsr2[15]&nfsr2[16] ⊕ nfsr2[25]&nfsr2[53] ⊕
86  nfsr2[35]&nfsr2[42] ⊕ nfsr2[55]&nfsr2[58] ⊕ nfsr2[60]&nfsr2[74]
87  ⊕ nfsr2[20]&nfsr2[22]&nfsr2[23] ⊕ nfsr2[62]&nfsr2[68]&nfsr2[72]
88  ⊕ nfsr2[77]&nfsr2[80]&nfsr2[81]&nfsr2[83]
89     for j = 0 to j <= 88 do
90        nfsr2[j] = nfsr2[j+1]
91     end for
92  end for
```

Tabel 5.16. (Lanjutan) *Pseudocode* implementasi algoritme Lizard.

```
93  fase pembentukan keystream
94  for t = 257 to t <= 473 do
95     nfsr1[30] = nfsr1[0] ⊕ nfsr1[5] ⊕ nfsr1[6] ⊕ nfsr1[15] ⊕
96  nfsr1[17] ⊕ nfsr1[18] ⊕ nfsr1[20] ⊕ nfsr1[25] ⊕
97  nfsr1[8]&nfsr1[18] ⊕ nfsr1[8]&nfsr1[20] ⊕ nfsr1[12]&nfsr1[21] ⊕
98  nfsr1[14]&nfsr1[19] ⊕ nfsr1[17]&nfsr1[21] ⊕ nfsr1[20]&nfsr1[22]
99  ⊕ nfsr1[4]&nfsr1[12]&nfsr1[22] ⊕ nfsr1[4]&nfsr1[19]&nfsr1[22] ⊕
100 nfsr1[7]&nfsr1[20]&nfsr1[21] ⊕ nfsr1[8]&nfsr1[18]&nfsr1[22] ⊕
101 nfsr1[8]&nfsr1[20]&nfsr1[22] ⊕ nfsr1[12]&nfsr1[19]&nfsr1[22] ⊕
102 nfsr1[20]&nfsr1[21]&nfsr1[22] ⊕
103 nfsr1[4]&nfsr1[7]&nfsr1[12]&nfsr1[21] ⊕
104 nfsr1[4]&nfsr1[7]&nfsr1[19]&nfsr1[21]⊕
105 nfsr1[4]&nfsr1[12]&nfsr1[21]&nfsr1[22] ⊕
106 nfsr1[4]&nfsr1[19]&nfsr1[21]&nfsr1[22] ⊕
107 nfsr1[7]&nfsr1[8]&nfsr1[18]&nfsr1[21] ⊕
108 nfsr1[7]&nfsr1[8]&nfsr1[20]&nfsr1[21] ⊕
109 nfsr1[7]&nfsr1[12]&nfsr1[19]&nfsr1[21] ⊕
110 nfsr1[8]&nfsr1[18]&nfsr1[21]&nfsr1[22] ⊕
111 nfsr1[8]&nfsr1[20]&nfsr1[21]&nfsr1[22] ⊕
112 nfsr1[12]&nfsr1[19]&nfsr1[21]&nfsr1[22]
113    for i = 0 to i <= 29 do
114       nfsr1[i] = nfsr1[i+1]
115    end for
116    nfsr2[89] = nfsr1[0] ⊕ nfsr2[0] ⊕ nfsr2[24] ⊕ nfsr2[49] ⊕
117 nfsr2[79] ⊕ nfsr2[84] ⊕ nfsr2[3]&nfsr2[59] ⊕ nfsr2[10]&nfsr2[12]
118 ⊕ nfsr2[15]&nfsr2[16] ⊕ nfsr2[25]&nfsr2[53] ⊕
119 nfsr2[35]&nfsr2[42] ⊕ nfsr2[55]&nfsr2[58] ⊕ nfsr2[60]&nfsr2[74]
120 ⊕ nfsr2[20]&nfsr2[22]&nfsr2[23] ⊕ nfsr2[62]&nfsr2[68]&nfsr2[72]
121 ⊕ nfsr2[77]&nfsr2[80]&nfsr2[81]&nfsr2[83]
122    for j = 0 to j <= 88 do
123       nfsr2[j] = nfsr2[j+1]
124    end for
125    keystream[i-257]= z
126 end for
```

Tabel 5.14 merupakan *pseudocode* dari implementasi algoritme Lizard pada perangkat mikrokontroler NodeMCU. Terdapat empat fase dalam proses inisialisasi *state* algoritme Lizard yang dilakukan dalam 257 kali siklus *clock* yaitu fase *key and iv loading*, *Grain-like mixing*, *second key add*, dan *final diffusion*. Fase pertama adalah *key and iv loading* digunakan untuk memuat kunci dan *IV* ke dalam NFSR1 dan NFSR2. Fase kedua dalam inisialisasi *state* adalah *Grain-like mixing* yang dijalankan pada siklus *clock* (t) 0 hingga 127. Fase tiga adalah *second key add* yang dijalankan pada *clock* ke 128. Fase keempat adalah *final diffusion*. Fase terakhir ini dijalankan pada *clock* ke 129 hingga 256. Setelah inisialisasi *state* selesai dijalankan, program akan memulai fase pembentukan *keystream* algoritme Lizard sebanyak 216 bit biner.

5.2.4 Implementasi Klien REST

Klien *web service* REST diimplementasikan pada perangkat mikrokontroler NodeMCU. Fitur-fitur yang diimplementasikan pada klien REST yaitu mengubah ASCII menjadi biner, mengenkripsi kumpulan bit biner menjadi *cipher text*, membaca nilai suhu dan kelembapan melalui sensor DHT11, serta melakukan *POST request* menuju server.

Tabel 5.17. *Pseudocode* implementasi fungsi mengubah ASCII ke bentuk biner.

```
1  fungsi mengubah ASCII ke bentuk biner
2  if ASCII = NULL return 0
3  for i = 0 to i < panjang ASCII do
4     karakter ← ASCII[i]
5     for j = 7 to j >= 0 do
6        if karakter and 1 << j do biner = "1"
7        else do biner = "0"
8     end for
9  end for
```

Tabel 5.17 berisi *pseudocode* yang menunjukkan fungsi untuk mengubah ASCII menjadi bentuk biner. JSON merupakan data masukkan pada fungsi mengubah ASCII ke bentuk biner dengan nilai keluaran yang dikembalikan adalah kumpulan bit biner. Setiap satu karakter dalam JSON akan diubah menjadi 8 bit biner. Salah satu contoh hasil masukan dan keluaran sebesar 27 bita dari fungsi mengubah ASCII ke bentuk biner dapat dilihat pada Tabel 5.18 di bawah.

Tabel 5.18. Nilai masukan dan keluaran dari fungsi mengubah ASCII ke bentuk biner.

Masukan	Keluaran

{
"t": 26,
"h": 55
}
	01111011 00001010 00100000
	00100000 00100010 01110100
	00100010 00111010 00100000
	00110010 00110110 00101100
	00001010 00100000 00100000
	00100010 01101000 00100010
	00111010 00100000 00110101
	00110101 00001010 01111101

Tabel 5.19. *Pseudocode* implementasi fungsi enkripsi.

```
1  fungsi enkripsi
2  for i = 0 to i <= panjang bit – 1 do
3     cipher[i] ← bit[i] ^ keystream[i]
4  end for
```

Tabel 5.19 berisi *pseudocode* yang menunjukkan fungsi untuk mengenkripsi kumpulan bit biner hasil keluaran dari fungsi mengubah ASCII ke bentuk biner. Kumpulan bit biner dari hasil keluaran tersebut merupakan data masukkan pada fungsi enkripsi dengan hasil keluaran adalah *cipher text* berbentuk kumpulan bit biner. Setiap bit biner akan di-*xor* dengan *keystream* dari algoritme Lizard sesuai urutannya, seperti bit biner *plain text* pertama dengan biner *keystream* pertama, biner *plain text* kedua dengan biner *keystream* kedua, dan seterusnya. Salah satu contoh hasil masukan dan keluaran sebesar 216 bit biner dari fungsi enkripsi dapat dilihat pada Tabel 5.20.

Tabel 5.20. Nilai masukan dan keluaran dari fungsi enkripsi.

Masukan	Keluaran
011110110000110100001010001000 000010000000100010011101000010 001000111010001000000011001000 110110001011000000110100001010 001000000010000000100010011010 000010001000111010001000000011 010100110101000011010000101001 111101	110011010011110101000110100001 001110101000000101000111110001 000101101111110011000001110000 100110101110101000001110001110 100100110001000101001100010011 100101001001100111111010101110 100001101000111010001010110111 110010

Tabel 5.21. *Pseudocode* implementasi proses pengiriman data menuju server.

```
1  aktifkan modul klien HTTP
2  kirim POST request ke server dengan url /post
3  atur HTTP header → "Content-Type", "text/plain")
4  kirim resource berisi cipher text
5  terima response dari server
```

Tabel 5.21 berisi *pseudocode* untuk proses pengiriman *POST request* beserta *resource* atau data menuju ke server REST. Pada baris pertama merupakan pendeklarasian diri sebagai HTTP klien. Pada baris 2 merupakan proses HTTP *request* ke alamat server dengan URL /post. *Resource* yang dikirim adalah cipher text yang berbentuk *string*. Kemudian, klien akan menerima *response* yang menunjukkan berhasil atau tidaknya data diproses oleh server. Tampilan sistem klien REST pada Serial Arduino IDE dapat dilihat pada Gambar 5.7 di bawah.

```
IP address: 192.168.1.10
{
  "t": 31,
  "h": 59
}
200
POSTED
IP address: 192.168.1.10
{
  "t": 30,
  "h": 58
}
200
POSTED
IP address: 192.168.1.10
{
  "t": 30,
  "h": 53
}
200
POSTED
```

Gambar 5.6. Tampilan hasil implementasi klien REST pada Arduino IDE.

5.2.5 Implementasi Server REST

Server *web service* REST diimplementasikan pada server basis data. Fitur-fitur yang diimplementasikan pada server REST yaitu menerima *request* HTTP, mengirimkan HTTP *response*, mendekripsi *cipher text* dari klien, mengubah biner menjadi bentuk ASCII, menyimpan nilai suhu dan kelembapan ke dalam basis data. Server REST dibangun dengan bahasa pemrograman Python.

Tabel 5.22. *Pseudocode* implementasi fungsi dekripsi.

```
1  fungsi dekripsi
2  i ← 0
3  while i < banyak bit cipher do
4     plaintext ← cipher[i] ^ keystream[i]
5     i += 1
```

Tabel 5.22 berisi *pseudocode* yang menunjukkan fungsi untuk mendekripsi *cipher text* atau *resource* yang diterima oleh server REST. *Cipher text* yeng berbentuk kumpulan bit biner tersebut akan di-*xor* dengan *keystream* dari algoritme Lizard sesuai urutannya, seperti bit biner *plain text* pertama dengan biner *keystream* pertama, biner *plain text* kedua dengan biner *keystream* kedua, dan seterusnya. Fungsi dekripsi menghasilkan *plain text* berbentuk biner. Salah satu contoh hasil masukan dan keluaran sebanyak 216 bit biner dari fungsi dekripsi dapat dilihat pada Tabel 5.23 di bawah.

Tabel 5.23. Nilai masukan dan keluaran dari fungsi dekripsi.

Masukan	Keluaran
110011010011110101000110100001 001110101000000101000111110001 000101101111110011000001110000 100110101110101000001110001110 100100110001000101001100010011 100101001001100111111010101110 100001101000111010001010110111 110010	011110110000110100001010001000 000010000000100010011101000010 001000111010001000000011001000 110110001011000000110100001010 001000000010000000100010011010 000010001000111010001000000011 010100110101000011010000101001 111101

Tabel 5.24. *Pseudocode* implementasi proses penerimaan dan penyimpanan data oleh server.

```
1  menerima request dan resource dengan URL /post
2  dekripsi resource
3  for i = 0 to i < banyak bit do
4     for j = 7 to j >= 0 do
5        ubah biner ke ASCII
6     end for
7  JSON loads
8  masukkan nilai suhu dan kelembapan ke basis data
```

Tabel 5.24 berisi *pseudocode* untuk menangani HTTP *request* dengan metode POST dan *URL* /post dari klien. Pada baris 2, dilakukan dekripsi pada data tersebut dengan memanggil fungsi `dekripsi` hingga menghasilkan *plain text* berbentuk kumpulan bit biner. Lalu, *plain text* tersebut diubah menjadi bentuk ASCII. Baris 7 menunjukkan bahwa sistem memanggil fungsi json.loads untuk mengeluarkan

tiap variabel suhu dan kelembapan dari dalam JSON. Selanjutnya, nilai suhu dan kelembapan disimpan dalam tabel basis data pada MySQL-Server. Tampilan sistem server REST saat berjalan dapat dilihat pada Gambar 5.8 di bawah.

Gambar 5.7. Tampilan hasil implementasi server REST pada *command prompt*.

5.2.6 Implementasi Basis data

Implementasi basis data pada sistem ini menggunakan aplikasi MySQL-Server. Terdapat empat kolom pembentuk basis data yaitu id, suhu, kelembapan, dan waktu. Kolom id menyimpan nomor identitas sebagai pembeda tiap data. Kolom suhu dan kelembapan menyimpan nilai suhu dan kelembapan yang telah diterima dan didekripsi oleh server REST. Kolom waktu menyimpan nilai *timestamp* saat data disimpan oleh server. Gambar 5.9 merupakan kumpulan data yang tersimpan pada basis data.

id ▲ 1	suhu	kelembaban	waktu
179	28	39	2018-05-02 16:58:36
180	28	39	2018-05-02 16:59:08
184	28	38	2018-05-02 17:34:43
185	28	34	2018-05-02 17:35:13
186	27	40	2018-05-02 17:36:19
187	27	40	2018-05-02 17:36:49

Gambar 5.8. Tampilan hasil penyimpanan data pada basis data MySQL-Server.

BAB 6 PENGUJIAN

Setelah proses implementasi selesai, langkah berikutnya yakni melakukan pengujian pada sistem. Pengujian ini dibutuhkan untuk mendapatkan data hasil uji serta untuk mengetahui apakah kinerja sistem yang telah dikembangkan sudah sesuai dengan kebutuhan dan perancangan yang dijelaskan pada bab-bab sebelumnya. Proses pengujian terbagi menjadi empat bagian yaitu pengujian fungsional, validasi *keystream* algoritme Lizard, pengujian kinerja, dan pengujian keamanan.

6.1 Pengujian Fungsional

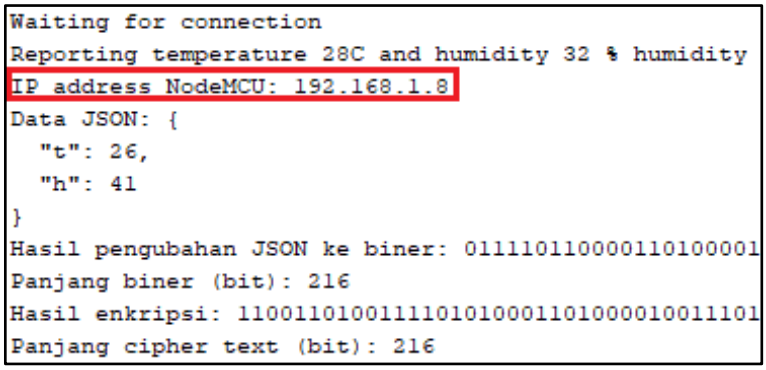
Pengujian fungsional dilakukan untuk melihat kesesuaian fungsi-fungsi hasil implementasi dengan perancangan. Dari hasil pengujian fungsional dapat dilihat apakah fungsi-fungsi pada sistem telah berjalan sesuai dengan perancangan.

6.1.1 Menyambungkan Perangkat Ke Jaringan *Wifi*

Kasus uji proses menyambungkan perangkat NodeMCU ke jaringan *wifi* dapat dilihat pada Tabel 6.1.

Tabel 6.1 Pengujian proses penyambungan perangkat ke jaringan *wifi*.

Kode	PF-01
Fungsi	Menyambungkan perangkat ke jaringan *wifi*
Tujuan pengujian	Menguji proses koneksi perangkat NodeMCU ke jaringan *wifi*.
Prosedur pengujian	1. Terdapat jaringan *wifi* yang aktif di sekitar perangkat NodeMCU. 2. Memasukkan nama *SSID* dan *password* jaringan *wifi*. 3. Menampilkan alamat *IP* (*Internet Protocol*) yang diperoleh.
Hasil yang diharapkan	Tersambung dengan jaringan *wifi* dan mendapatkan alamat *IP*.
Hasil pengujian	Berhasil tersambung dengan jaringan *wifi* dan mendapatkan alamat *IP*.



Gambar 6.1. Tampilan hasil pengujian PF-01 pada Arduino IDE.

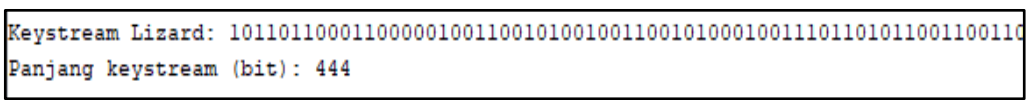
Pertama, NodeMCU menjalankan fungsi WiFi.begin("MB 39", "shameonyou") yang mana MB 39 dan shameonyou merupakan *SSID* dan *password* dari perangkat router yang memiliki jaringan 192.168.1.0/24. Setelah perangkat NodeMCU terhubung dengan jaringan *wifi*, perangkat mendapatkan alamat *IP* 192.168.1.8. Gambar 6.1 merupakan alamat *IP* NodeMCU yang ditampilkan pada serial Arduino IDE.

6.1.2 Membentuk *Keystream* Algoritme Lizard

Kasus uji proses pembentukan *keystream* algoritme Lizard dapat dilihat pada Tabel 6.2.

Tabel 6.2. Pengujian proses pembentukan *keystream* algoritme Lizard

Kode	PF-02
Fungsi	Membentuk *keystream* algoritme Lizard
Tujuan pengujian	Menguji proses pembentukan *keystream* algoritme Lizard sebesar 444 bit.
Prosedur pengujian	1. Perangkat NodeMCU tersambung dengan jaringan *wifi*. 2. Deklarasi semua bit *key* dan *IV* algoritme Lizard adalah 0. 3. Menampilkan hasil pembentukan *keystream*.
Hasil yang diharapkan	Terbentuk *keystream* algoritme Lizard sebesar 444 bit.
Hasil pengujian	Berhasil terbentuk *keystream* algoritme Lizard sebesar 444 bit.



Gambar 6.2. Tampilan hasil pengujian PF-02 pada Arduino IDE.

Gambar 6.2 menunjukkan *keystream* algoritme Lizard sebesar 444 bit yang ditampilkan pada serial Arduino IDE. *Keystream* tersebut dibentuk dari kombinasi *key* 120 bit dan *IV* 64 bit yang semua bit binernya adalah 0.

6.1.3 Membaca Nilai Suhu Dan Kelembapan

Kasus uji proses pembacaan nilai suhu dan kelembapan lingkungan sekitar dapat dilihat pada Tabel 6.3.

Tabel 6.3. Pengujian proses pembacaan nilai suhu dan kelembapan.

Kode	PF-03
Fungsi	Membaca nilai suhu dan kelembapan
Tujuan pengujian	Menguji proses pembacaan nilai suhu dan kelembapan lingkungan melalui sensor DHT11.
Prosedur pengujian	1. Perangkat NodeMCU tersambung dengan jaringan *wifi*. 2. Sensor DHT11 tersambung dengan mikrokontroler NodeMCU. 3. Menampilkan nilai suhu dan kelembapan.
Hasil yang diharapkan	Membaca nilai suhu dan kelembapan lingkungan sekitar sensor dan mikrokontroler berada.
Hasil pengujian	Berhasil membaca nilai suhu dan kelembapan lingkungan sekitar sensor dan mikrokontroler berada.

```
Waiting for connection
Reporting temperature 28C and humidity 32 % humidity
IP address NodeMCU: 192.168.1.8
Data JSON: {
  "t": 26,
  "h": 41
}
Hasil pengubahan JSON ke biner: 01111011000011010000l
Panjang biner (bit): 216
Hasil enkripsi: 11001101001111010100011010000100111101
Panjang cipher text (bit): 216
```

Gambar 6.3. Tampilan hasil pengujian PF-03 pada Arduino IDE.

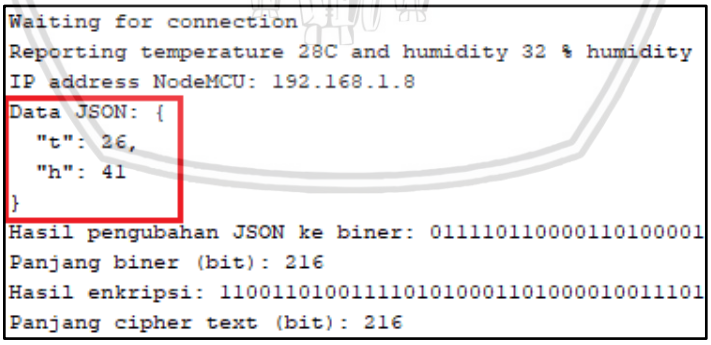
Gambar 6.3 menunjukkan nilai suhu dan kelembapan dari lingkungan sekitar sensor DHT11. Sensor berhasil menangkap nilai 26 derajat celcius dan 41% kelembapan.

6.1.4 Membungkus Data Suhu Dan Kelembapan Ke Dalam JSON

Kasus uji proses pembungkusan data suhu dan kelembapan ke dalam JSON dapat dilihat pada Tabel 6.4.

Tabel 6.4. Pengujian proses pembungkusan data suhu dan kelembapan.

Kode	PF-04
Fungsi	Membungkus data suhu dan kelembapan ke dalam JSON
Tujuan pengujian	Menguji pembungkusan daya suhu dan kelembapan ke dalam JSON dengan *key* 'h' untuk kelembapan dan 't' untuk suhu.
Prosedur pengujian	1. Perangkat NodeMCU tersambung dengan jaringan *wifi*. 2. Sensor DHT11 tersambung dengan mikrokontroler NodeMCU. 3. Membungkus nilai suhu dengan *key* 't' dan kelembapan dengan *key* 'h' ke dalam JSON. 4. Menampilkan data JSON.
Hasil yang diharapkan	Membungkus nilai suhu dan kelembapan ke dalam JSON.
Hasil pengujian	Berhasil membungkus nilai suhu dan kelembapan ke dalam JSON.



Gambar 6.4. Tampilan hasil pengujian PF-04 pada Arduino IDE.

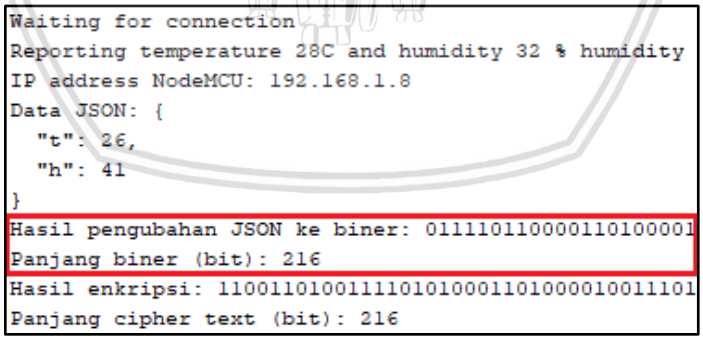
Gambar 6.4 menunjukkan JSON hasil pembungkusan data suhu senilai 26 dan kelembapan senilai 41 yang telah dibaca oleh sensor. JSON terdiri dari *key* "t" dan "h" yang menyimpan nilai suhu dan kelembapan.

6.1.5 Mengubah Data JSON Menjadi Bentuk Biner

Kasus uji proses pengubahan JSON menjadi bentuk biner dapat dilihat pada Tabel 6.5.

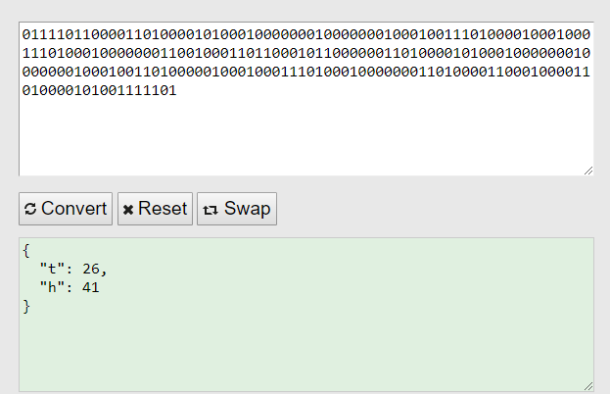
Tabel 6.5. Pengujian proses pengubahan JSON menjadi bentuk biner.

Kode	PF-05
Fungsi	Mengubah JSON menjadi bentuk biner
Tujuan pengujian	Menguji proses pengubahan JSON menjadi bentuk biner.
Prosedur pengujian	1. Perangkat NodeMCU tersambung dengan jaringan *wifi*. 2. Sensor DHT11 tersambung dengan mikrokontroler NodeMCU. 3. Memasukkan JSON ke dalam fungsi stringToBinary(). 4. Menampilkan hasil pengubahan JSON menjadi bentuk biner dan memvalidasi biner dengan bantuan aplikasi rapidtables.com.
Hasil yang diharapkan	Mengubah setiap karakter ASCII dalam JSON menjadi bentuk biner.
Hasil pengujian	Berhasil mengubah setiap karakter ASCII dalam JSON menjadi bentuk biner.



Gambar 6.5. Tampilan hasil pengujian PF-05 pada Arduino IDE.

Gambar 6.5 menunjukkan hasil pengubahan JSON menjadi bentuk biner sebesar 216 bit. JSON dimasukkan ke dalam fungsi stringToBinary() yang mengembalikan nilai *plain text* berbentuk biner.



Gambar 6.6. Tampilan pengubahan *plain text* biner ke bentuk ASCII pada rapidtables.com.

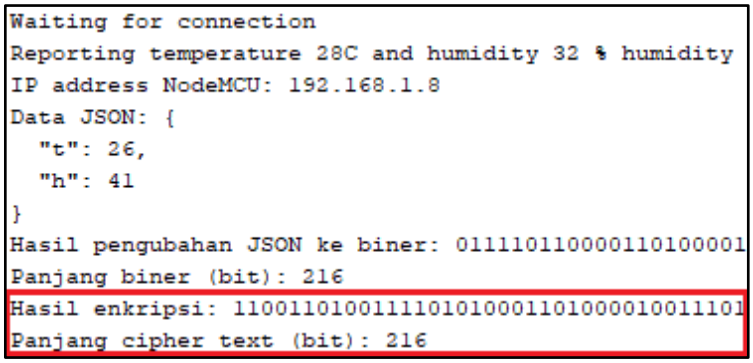
Gambar 6.6 menunjukkan pengubahan biner kembali ke bentuk ASCII dengan bantuan aplikasi rapidtables.com. Proses pengubahan menunjukkan hasil ASCII yang identik dengan JSON awal. Pengujian ini berguna untuk memastikan bahwa fungsi stringToBinary() telah berjalan dengan baik.

6.1.6 Mengenkripsi *Plain Text* Menjadi *Cipher Text*

Kasus uji proses enkripsi *plain text* berbentuk biner menjadi *cipher text* dapat dilihat pada Tabel 6.6.

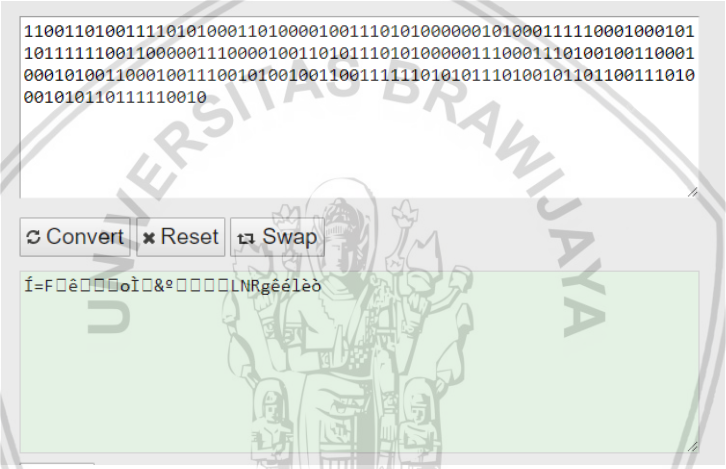
Tabel 6.6. Pengujian proses enkripsi *plain text*.

Kode	PF-06
Fungsi	Mengenkripsi *plain text* menjadi *cipher text*
Tujuan pengujian	Menguji proses enkripsi *plain text* menjadi *cipher text* dengan algoritme Lizard.
Prosedur pengujian	1. Perangkat NodeMCU tersambung dengan jaringan *wifi*. 2. Sensor DHT11 tersambung dengan mikrokontroler NodeMCU. 3. *Keystream* berhasil dibentuk. 4. Berhasil mengubah JSON menjadi bentuk biner. 5. Memasukkan biner hasil pengubahan JSON ke dalam fungsi enkripsi_json(). 6. Menampilkan hasil enkripsi dan memvalidasi biner dengan bantuan aplikasi rapidtables.com.
Hasil yang diharapkan	Mengubah *plain text* menjadi *cipher text* yang tidak dapat terbaca.
Hasil pengujian	Berhasil mengubah *plain text* menjadi *cipher text* yang tidak dapat terbaca.



Gambar 6.7. Tampilan hasil pengujian PF-06 pada Arduino IDE.

Gambar 6.7 menunjukkan hasil enkripsi *plain text* menjadi *cipher text* berbentuk biner sebesar 216 bit. *Plain text* berbentuk biner dimasukkan ke dalam fungsi enkripsi_json() yang mengembalikan nilai *cipher text* bertipe *string*.



Gambar 6.8. Tampilan pengubahan *cipher text* biner ke bentuk ASCII pada rapidtables.com

Gambar 6.8 menunjukkan pengubahan biner ke bentuk ASCII dengan bantuan aplikasi rapidtables.com. Proses pengubahan menunjukkan hasil ASCII yang tidak dapat terbaca (*cipher text*). Pengujian ini berguna untuk memastikan bahwa fungsi enkripsi_json() telah berjalan dengan baik.

6.1.7 Mengirim HTTP *Request* Dan *Cipher Text* Menuju Server REST

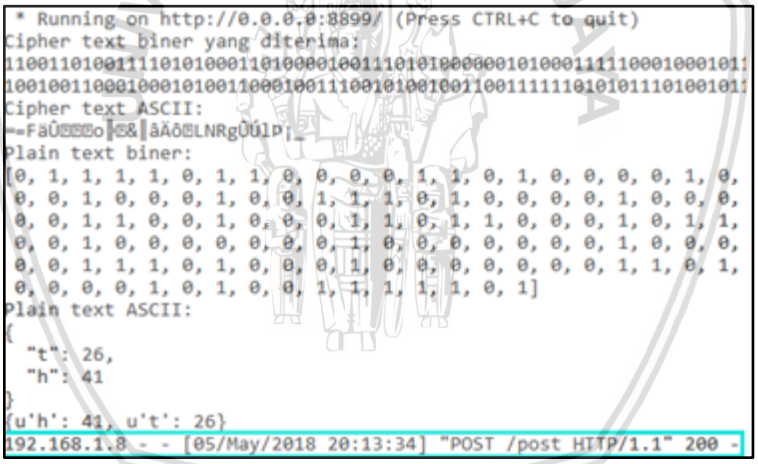
Kasus uji proses pengiriman HTTP *request* beserta *cipher text* dari klien menuju ke server REST dapat dilihat pada Tabel 6.7.

Tabel 6.7. Pengujian proses pengiriman HTTP *request* dan *cipher text*.

Kode	PF-07
Fungsi	Mengirim HTTP *request* dan *cipher text menuju* server
Tujuan pengujian	Menguji proses pengiriman HTTP *request* dan *cipher text* dari klien REST menuju server.

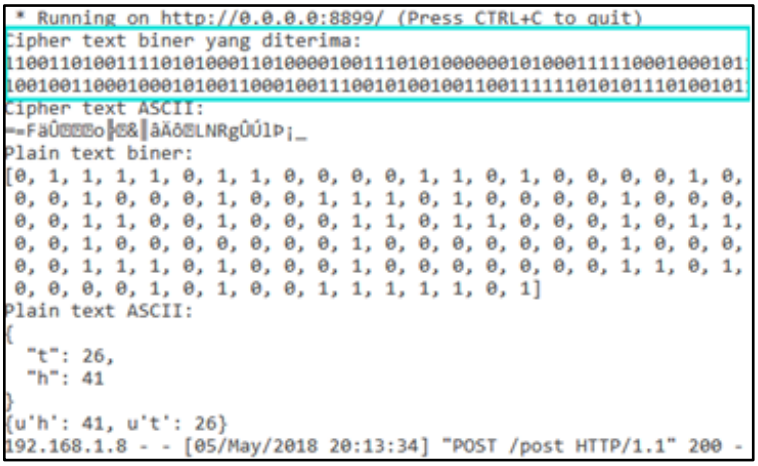
Table 6.8. (Lanjutan) Pengujian proses pengiriman HTTP *request* dan *cipher text*.

Prosedur pengujian	1. Perangkat NodeMCU tersambung dengan jaringan *wifi*. 2. Sensor DHT11 tersambung dengan mikrokontroler NodeMCU. 3. Berhasil mengenkripsi *plain text* menjadi *cipher text*. 4. Mengirim HTTP *request* dan *cipher text* ke alamat *IP* server REST 192.168.1.X:8899/post. 5. Melihat hasil pengiriman pada server REST tujuan.
Hasil yang diharapkan	Mengirim HTTP *request* dan *cipher text* hingga sampai ke server REST tujuan.
Hasil pengujian	Berhasil mengirim HTTP *request* dan *cipher text* hingga sampai ke server REST tujuan.



Gambar 6.9. Tampilan hasil pengiriman HTTP *request* pada *command prompt*.

Gambar 6.9 menunjukkan HTTP *request* yang dikirimkan oleh klien dengan alamat *IP* 192.168.1.8 menuju server REST. Klien berhasil mengirim HTTP *POST request* dengan *URL* /post menuju ke server REST.



Gambar 6.9. Tampilan hasil pengiriman *cipher text* pada *command prompt*.

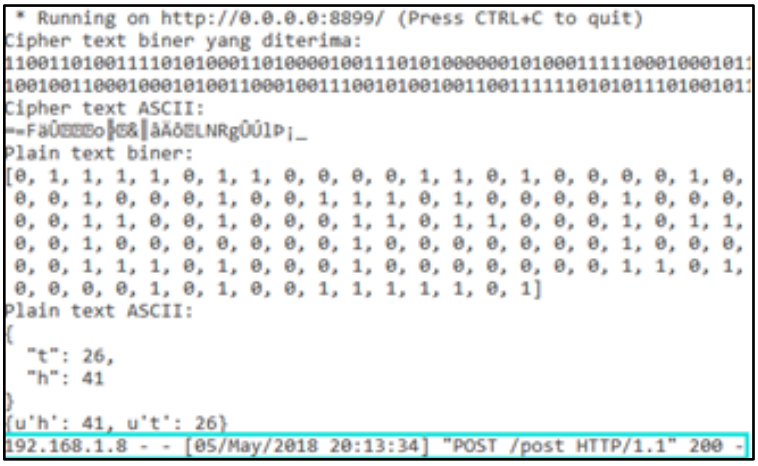
Gambar 6.10 menunjukkan *cipher text* sebesar 216 bit yang dikirimkan oleh klien menuju ke server. *Cipher text* dikirim bersamaan dengan HTTP *request*.

6.1.8 Menerima HTTP *Request* Dan *Cipher Text* Dari Klien

Kasus uji proses penerimaan HTTP *request* dan *cipher text* berbentuk biner dari klien dapat dilihat pada Tabel 6.9.

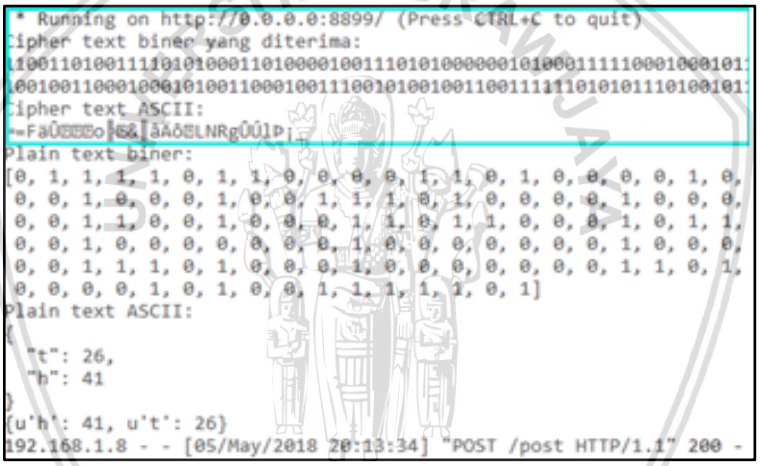
Tabel 6.9. Pengujian proses penerimaan HTTP *request* dan *cipher text*.

Kode	PF-08
Fungsi	Menerima HTTP *request* dan *cipher text* dari klien
Tujuan pengujian	Menguji proses penerimaan HTTP *request* dan *cipher text* dari klien.
Prosedur pengujian	1. Aplikasi basis data aktif. 2. Server REST aktif. 3. Menampilkan alamat *IP* klien dan *cipher text* yang diterima.
Hasil yang diharapkan	Menerima HTTP *request* dan *cipher text* dari klien.
Hasil pengujian	Berhasil menerima HTTP *request* dan *cipher text* dari klien.



Gambar 6.10. Tampilan hasil penerimaan HTTP *request* pada *command prompt*.

Gambar 6.11 menunjukkan HTTP *request* yang diterima oleh server dari klien dengan alamat *IP* 192.168.1.8. Server REST berhasil menerima HTTP *POST request* dengan *URL* /post menuju ke server REST.



Gambar 6.11. Tampilan hasil penerimaan *cipher text* pada *command prompt*.

Gambar 6.12 menunjukkan *cipher text* sebesar 216 bit yang diterima oleh server. *Cipher text* diterima bersamaan dengan masuknya HTTP *request*.

6.1.9 Mendekripsi *Cipher Text* Menjadi *Plain Text*

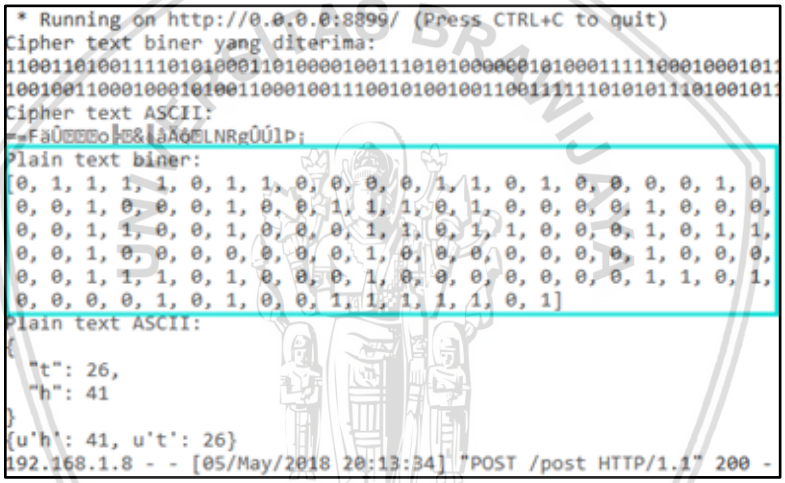
Kasus uji proses dekripsi *cipher text* menjadi *plain text* berbentuk biner dapat dilihat pada Tabel 6.10.

Tabel 6.10. Pengujian proses dekripsi *cipher text*.

Kode	PF-09
Fungsi	Mendekripsi *cipher text* menjadi *plain text*
Tujuan pengujian	Menguji proses dekripsi *cipher text* yang diterima dari klien menjadi *plain text*.

Table 6.11. (Lanjutan) Pengujian proses dekripsi *cipher text*.

Prosedur pengujian	1. Aplikasi basis data aktif. 2. Server REST aktif. 3. Berhasil menerima HTTP *request* dan *cipher text*. 4. Memasukkan *cipher text* yang diterima ke dalam fungsi decrypt(). 5. Menampilkan hasil dekripsi.
Hasil yang diharapkan	Mengubah *cipher text* menjadi *plain text* yang dapat dibaca.
Hasil pengujian	Berhasil mengubah *cipher text* menjadi *plain text* yang dapat dibaca.



Gambar 6.12. Tampilan hasil pengujian PF-09 pada *command prompt*.

Gambar 6.13 menunjukkan hasil dekripsi *cipher text* menjadi *plain text* berbentuk biner sebesar 216 bit. *Cipher text* berbentuk biner didekripsi dengan *keystream* yang sama dengan saat proses enkripsi di sisi klien. Fungsi decrypt() menghasilkan *plain text* biner yang identik dengan *plain text* pada sisi klien.

6.1.10 Mengubah Data Berbentuk Biner Menjadi ASCII

Kasus uji proses pengubahan *plain text* berbentuk biner menjadi bentuk ASCII dapat dilihat pada Tabel 6.12.

Tabel 6.12. Pengujian proses pengubahan *plain text* biner menjadi ASCII.

Kode	PF-10
Fungsi	Mengubah kumpulan bit biner menjadi bentuk ASCII
Tujuan pengujian	Menguji proses pengubahan data *plain text* berbentuk biner menjadi ASCII.

Table 6.13. (Lanjutan) Pengujian proses pengubahan *plain text* biner menjadi ASCII.

Prosedur pengujian	1. Aplikasi basis data aktif. 2. Server REST aktif. 3. Berhasil menerima HTTP *request* dan *cipher text*. 4. Berhasil melakukan proses dekripsi. 5. Memasukkan biner hasil dekripsi ke dalam proses pengubahan bentuk dari biner ke ASCII. 6. Menampilkan hasil pengubahan biner menjadi bentuk ASCII. 7. Membandingkan 3 data *plain text* ASCII hasil dekripsi dengan *plain text* JSON pada sisi klien.
Hasil yang diharapkan	Mengubah data berbentuk biner menjadi bentuk ASCII.
Hasil pengujian	Berhasil mengubah data berbentuk biner menjadi bentuk ASCII.

Setelah prosedur pengujian selesai dijalankan, ditemukan bahwa *plain text* di sisi klien memiliki bentuk yang identik dengan *plain text* hasil dekripsi pada sisi server. Berdasarkan hasil pengujian tersebut, dapat dikatakan bahwa data yang dikirim oleh klien dan diterima oleh server REST adalah identik. Hasil pengujian dapat dilihat pada Tabel 6.14 di bawah.

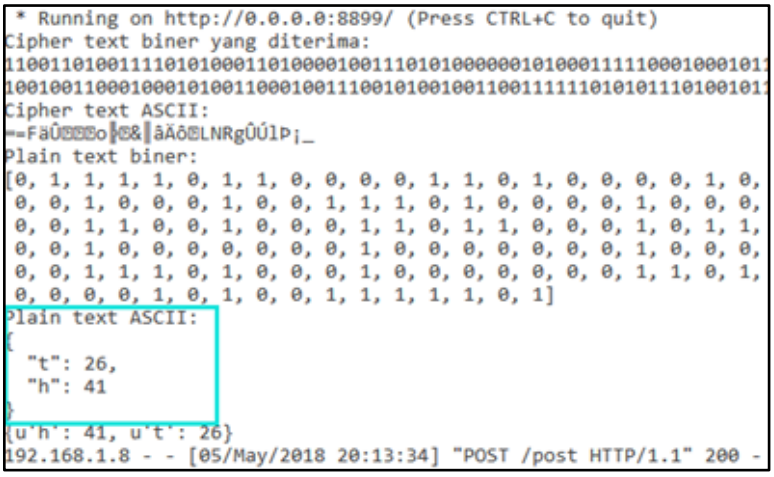
Tabel 6.14. Pengujian validitas *plain text* hasil dekripsi.

No	*Plain text* ASCII di sisi *klien*	*Plain text* ASCII di sisi server	Status

1	{
	"t": 26,
	"h": 41
	}
		{
		"t": 26,
		"h": 41
		}
			Valid

2	{
	"t": 30,
	"h": 58
	}
		{
		"t": 30,
		"h": 58
		}
			Valid

3	{
	"t": 30,
	"h": 53
	}
		{
		"t": 30,
		"h": 53
		}
			Valid



Gambar 6.13. Tampilan hasil pengujian PF-10 pada *command prompt*.

Gambar 6.14 menunjukkan JSON hasil pengubahan biner menjadi bentuk ASCII. Fungsi pengubahan bentuk biner ke ASCII menghasilkan JSON yang identik dengan *plain text* JSON pada sisi klien.

6.1.11 Menyimpan Data Ke Dalam Basis Data

Kasus uji proses penyimpanan data ke dalam basis data MySQL-Server dapat dilihat pada Tabel 6.15.

Tabel 6.15. Pengujian proses penyimpanan data.

Kode	PF-11
Fungsi	Menyimpan data ke dalam basis data
Tujuan pengujian	Menguji proses penyimpan nilai suhu dan kelembapan ke dalam basis data.
Prosedur pengujian	1. Aplikasi basis data aktif. 2. Server REST aktif. 3. Berhasil menerima HTTP *request* dan *cipher text*. 4. Berhasil mengubah biner menjadi bentuk ASCII. 5. Melihat data hasil penyimpanan pada basis data melalui aplikasi PHPMyAdmin.
Hasil yang diharapkan	Menyimpan nilai suhu dan kelembapan yang diterima oleh server ke dalam basis data.
Hasil pengujian	Berhasil menyimpan nilai suhu dan kelembapan yang diterima oleh server ke dalam basis data.

id ▲ 1	suhu	kelembaban	waktu
278	26	41	2018-05-05 20:13:33
279	26	41	2018-05-05 20:14:04
280	26	42	2018-05-05 20:14:35
281	26	42	2018-05-05 20:15:06
282	26	42	2018-05-05 20:15:36

Gambar 6.14. Tampilan hasil pengujian PF-11 pada basis data MySQL-Server.

Gambar 6.15 menunjukkan data yang berhasil dimasukkan ke dalam basis data MySQL-Server. Terdapat empat kolom yang menyimpan nilai id, suhu, kelembapan, dan waktu data diterima oleh server. Nilai dari masing-masing kolom dijabarkan pada Tabel 6.16 di bawah.

Tabel 6.16. Isi kolom pada tabel basis data.

id	suhu	kelembapan	Waktu
278	26	41	2018-05-05 20:13:33

6.2 Pengujian Validasi *Keystream* Algoritme Lizard

Pengujian ini bertujuan memastikan algoritme kriptografi Lizard yang telah diimplementasikan memiliki keluaran *keystream* yang sesuai dengan algoritme Lizard yang telah dibuat oleh penciptanya. Kasus uji validasi *keystream* algoritme Lizard dapat dilihat pada Tabel 6.17.

Tabel 6.17. Pengujian validasi *keystream* algoritme Lizard.

Kode	PK-01
Nama uji	Pengujian validasi *keystream*
Tujuan pengujian	Menguji kesamaan *keystream* algoritme Lizard dengan *test vector* yang dibuat oleh Hamann.

Table 6.18. (Lanjutan) Pengujian validasi *keystream* algoritme Lizard.

Prosedur pengujian	1. Membentuk *keystream* algoritme Lizard sebesar 128 bit dari kombinasi kunci 0x00000000000000000000000000000000 dan *IV* 0x0000000000000000. 2. Membentuk *keystream* algoritme Lizard sebesar 128 bit dari kombinasi kunci 0x00000000000000000FFFFFFFFFFFFFFF dan *IV* 0xFFFFFFFFFFFFFFFF. 3. Membentuk *keystream* algoritme Lizard sebesar 128 bit dari kombinasi kunci 0x0123456789ABCDEF0123456789ABCD dan *IV* 0xABCDEF0123456789. 4. Mengubah hasil *keystream* biner dari setiap kombinasi ke dalam bentuk heksadesimal dengan menggunakan aplikasi rapidtables.com. 5. Membandingkan hasil pembentukan 128 bit *keystream* algoritme Lizard pada perangkat NodeMCU dengan *test vector*.
Hasil yang diharapkan	Proses pembentukan 128 bit *keystream* pada NodeMCU memiliki hasil yang identik dengan *test vector*.
Hasil pengujian	128 bit *keystream* pada NodeMCU identik dengan *test vector*.

Setelah prosedur pengujian selesai dijalankan, ditemukan bahwa *keystream* yang digunakan dalam penelitian ini memiliki hasil yang identik dengan *keystream* pada *test vector*. Berdasarkan hasil pengujian tersebut, dapat dikatakan bahwa algoritme Lizard yang telah diimplementasi bersifat valid. Hasil pengujian dapat dilihat pada Tabel 6.19 di bawah.

Tabel 6.19. Hasil pengujian validasi *keystream* algoritme Lizard.

Kombinasi kunci dan *IV*	***Test vector***	**Hasil pembentukan *keystream***	**Status**
Kunci: 0x00000000000000000000000000000000 *IV*: 0x0000000000000000	*Keystream:* 0xB6304CA4CA276B3355EC2E10968E84B3	*Keystream:* 0xB6304CA4CA276B3355EC2E10968E84B3	Valid

Table 6.20. (Lanjutan) Hasil pengujian validasi *keystream* algoritme Lizard.

Kombinasi kunci dan *IV*	***Test vector***	**Hasil pembentukan *keystream***	**Status**
Kunci: 0x00000000000000FFFFFFFFFFFFFF *IV*: 0xFFFFFFFFFFFFFFFF	*Keystream:* 0x4D190941816F942358F0D164F4ECEB09	*Keystream:* 0x4D190941816F942358F0D164F4ECEB09	Valid
Kunci: 0x0123456789ABCDEF0123456789ABCD *IV*: 0xABCDEF0123456789	*Keystream:* 0x983311A97831586548209DAFBF26FC93	*Keystream:* 0x983311A97831586548209DAFBF26FC93	Valid

6.3 Pengujian Kinerja

Pengujian kinerja dibagi menjadi dua bagian, yaitu pengujian waktu dan memori pada proses pembentukan *keystream* dan pengujian waktu dan memori pada proses enkripsi.

6.3.1 Waktu Dan Memori Pada Proses Pembentukan *Keystream*

Kasus uji kinerja penggunaan waktu dan memori yang dibutuhkan dalam proses pembentukan *keystream* pada perangkat mikrokontroler NodeMCU dapat dilihat pada Tabel 6.21. Pengujian 216 dan 352 bit *keystream* dilakukan agar sama seperti panjang data yang dienkripsi.

Tabel 6.21. Pengujian kinerja perangkat NodeMCU pada proses pembentukan *keystream* algoritme Lizard.

Kode	PK-01
Nama uji	Pengujian waktu dan memori pada proses pembentukan *keystream*
Tujuan pengujian	Menguji waktu serta memori yang diperlukan dalam proses pembentukan *keystream*.

Table 6.22. (Lanjutan) Pengujian kinerja perangkat NodeMCU pada proses pembentukan *keystream* algoritme Lizard.

Prosedur pengujian	1. Melihat waktu yang diperlukan oleh perangkat NodeMCU untuk membentuk 216 dan 352 bit *keystream* pada tiap skenario dengan menggunakan fungsi micros(). 2. Melihat memori yang diperlukan oleh perangkat NodeMCU untuk membentuk 216 dan 352 bit *keystream* pada tiap skenario dengan menggunakan fungsi ESP.getFreeHeap(). 3. Pengujian pada skenario ini akan dilakukan sebanyak 50 kali proses dan hasilnya akan dirata-rata.
Skenario pertama	120 bit kunci = 0 64 bit *IV* = 0
Skenario kedua	120 bit kunci = 0 64 bit *IV* = 1
Skenario ketiga	120 bit kunci = 1 64 bit *IV* = 0
Skenario keempat	120 bit kunci = 1 64 bit *IV* = 1
Skenario kelima	60 bit kunci awal (indeks 1-60) = 0 60 bit kunci akhir (indeks 61-120) = 1 32 bit *IV* awal (indeks 1-32) = 0 32 bit *IV* akhir (indeks 33-64) = 1
Skenario keenam	60 bit kunci awal (indeks 1-60) = 0 60 bit kunci akhir (indeks 61-120) = 1 32 bit *IV* awal (indeks 1-32) = 1 32 bit *IV* akhir (indeks 33-64) = 0
Skenario ketujuh	60 bit kunci awal (indeks 1-60) = 1 60 bit kunci akhir (indeks 61-120) = 0 32 bit *IV* awal (indeks 1-32) = 0 32 bit *IV* akhir (indeks 33-64) = 1

Table 6.23. (Lanjutan) Pengujian kinerja perangkat NodeMCU pada proses pembentukan *keystream* algoritme Lizard.

Skenario kedelapan	60 bit kunci awal (indeks 1-60) = 1 60 bit kunci akhir (indeks 61-120) = 0 32 bit *IV* awal (indeks 1-32) = 1 32 bit *IV* akhir (indeks 33-64) = 0

Tabel 6.24 menunjukkan hasil pengujian waktu dan memori yang dibutuhkan oleh perangkat mikrokontroler NodeMCU saat membentuk *keystream* algoritme Lizard sebesar 216 bit. Proses pembentukan *keystream* sebesar 216 bit pada setiap skenario membutuhkan waktu ± 10200 mikrodetik atau 0,0102 detik. Hasil pengujian menunjukkan bahwa skenario ketiga membutuhkan waktu paling lama yaitu 10228,61 mikrodetik dan skenario pertama membutuhkan waktu paling cepat yaitu 10189,05 mikrodetik. Memori yang dibutuhkan oleh perangkat mikrokontroler NodeMCU untuk membentuk 216 bit *keystream* pada setiap skenario adalah sama yaitu 2720 bita.

Tabel 6.24. Hasil pengujian waktu dan memori pembentukan *keystream* 216 bit.

	Waktu (mikrodetik)	Memori (bita)
Skenario 1	10189,05	2720
Skenario 2	10206,96	
Skenario 3	10228,61	
Skenario 4	10209,55	
Skenario 5	10224,33	
Skenario 6	10225,76	
Skenario 7	10212,29	
Skenario 8	10204,48	

Tabel 6.25 menunjukkan hasil pengujian waktu dan memori yang dibutuhkan oleh perangkat mikrokontroler NodeMCU saat membentuk *keystream* algoritme Lizard sebesar 352 bit. Proses pembentukan *keystream* sebesar 352 bit pada setiap skenario membutuhkan waktu ± 13200 mikrodetik atau 0,0132 detik. Hasil pengujian menunjukkan bahwa skenario ketiga membutuhkan waktu paling lama yaitu 13289,71 mikrodetik dan skenario pertama membutuhkan waktu paling cepat yaitu 13251,77 mikrodetik. Memori yang dibutuhkan oleh perangkat mikrokontroler NodeMCU untuk membentuk 352 bit *keystream* pada setiap skenario adalah sama yaitu 2864 bita.

Tabel 6.25. Hasil pengujian waktu dan memori pembentukan *keystream* 352 bit.

	Waktu (mikrodetik)	Memori (bita)
Skenario 1	13251,77	2864
Skenario 2	13273,04	
Skenario 3	13289,71	
Skenario 4	13267,32	
Skenario 5	13283,69	
Skenario 6	13287,32	
Skenario 7	13275,71	
Skenario 8	13260,04	

6.3.2 Waktu Dan Memori Pada Proses Enkripsi

Kasus uji kinerja penggunaan waktu dan memori yang dibutuhkan dalam proses enkripsi data pada perangkat mikrokontroler NodeMCU dapat dilihat pada Tabel 6.26.

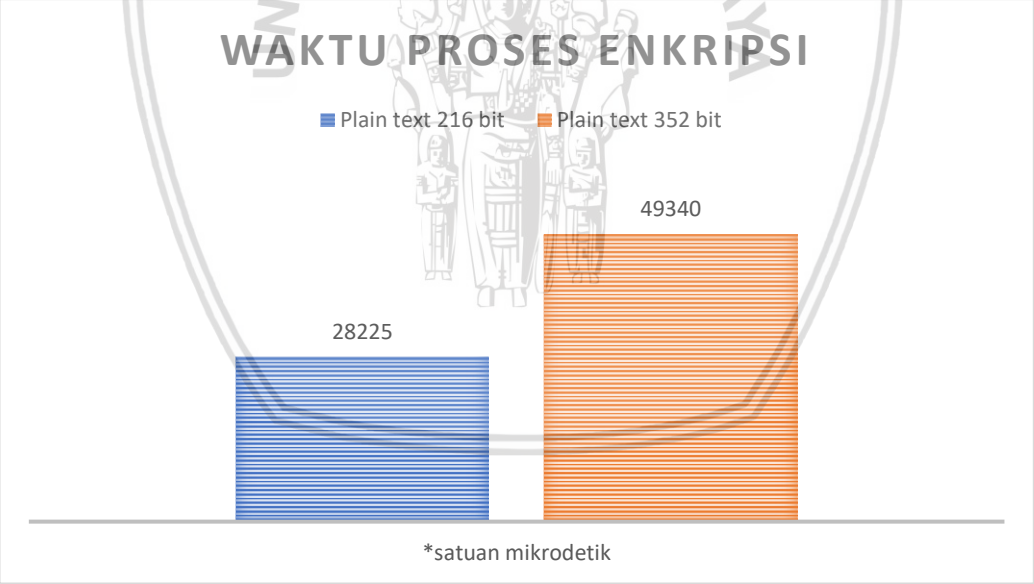
Tabel 6.26. Pengujian kinerja perangkat NodeMCU pada proses enkripsi data.

Kode	PK-02
Nama uji	Pengujian waktu dan memori pada proses enkripsi data
Tujuan pengujian	Menguji dan menganalisa waktu serta memori yang diperlukan dalam proses enkripsi *plain text* hingga menjadi *cipher text*.
Prosedur pengujian	1. Melihat waktu yang diperlukan oleh perangkat NodeMCU untuk melakukan proses enkripsi *plain text* pada tiap skenario dengan menggunakan fungsi micros(). 2. Melihat memori yang diperlukan oleh perangkat NodeMCU melakukan proses enkripsi *plain text* pada tiap skenario dengan menggunakan fungsi ESP.getFreeHeap(). 3. Pengujian pada skenario ini akan dilakukan sebanyak 50 kali proses secara berturut-turut dan hasilnya akan dirata-rata.

Table 6.27. (Lanjutan) Pengujian kinerja perangkat NodeMCU pada proses enkripsi data.

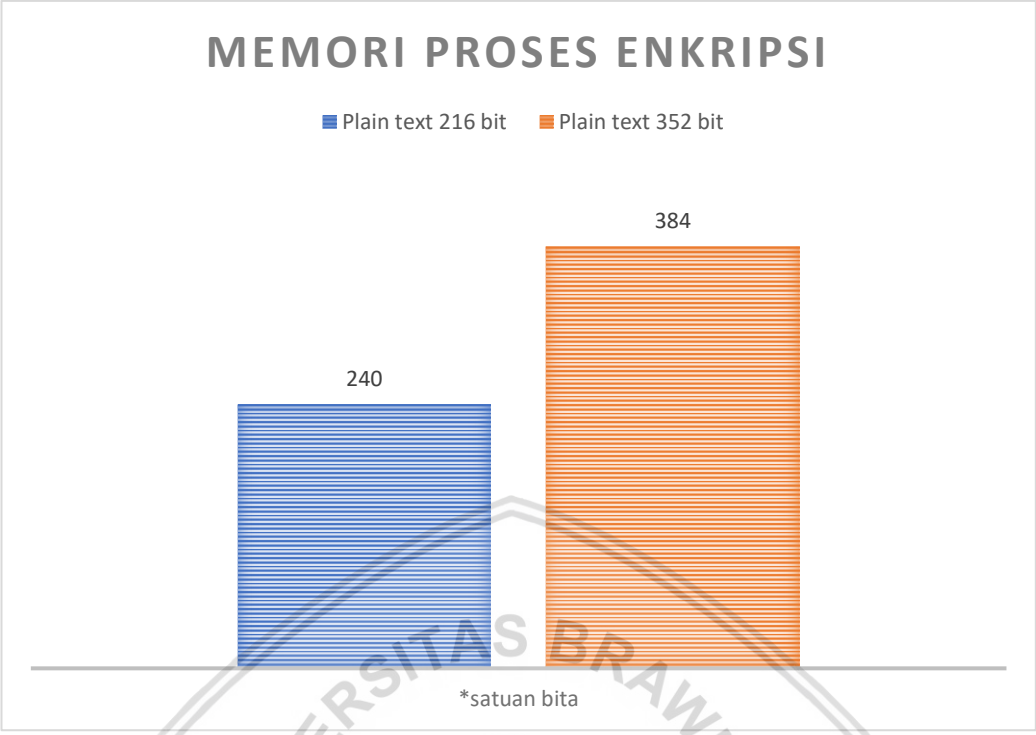
Skenario pertama	Enkripsi *plain text* sebesar 216 bit.
	Plain text:
	{
	"t": 23,
	"h": 51
	}

Skenario kedua	Enkripsi *plain text* sebesar 352 bit.
	Plain text:
	{
	"temperature": 23,
	"humidity": 51
	}



Gambar 6.16. Grafik waktu enkripsi data pada perangkat NodeMCU.

Gambar 6.16 merupakan grafik hasil pengujian waktu yang dibutuhkan oleh perangkat mikrokontroler NodeMCU saat mengenkripsi *plain text* sebesar 216 bit dan 352 bit. Proses enkripsi *plain text* sebesar 216 bit membutuhkan waktu 28225 mikrodetik atau 0,028225 detik. Proses enkripsi *plain text* sebesar 352 bit membutuhkan waktu 49340 mikrodetik atau 0,049340 detik. Proses enkripsi *plain text* 216 bit dengan 352 bit memiliki selisih waktu 21.115 mikrodetik atau mengalami kenaikan 74,8%.



Gambar 6.17. Grafik memori enkripsi data pada perangkat NodeMCU.

Gambar 6.17 merupakan grafik hasil pengujian memori yang dibutuhkan oleh perangkat mikrokontroler NodeMCU saat mengenkripsi *plain text* sebesar 216 bit dan 328 bit. Proses enkripsi *plain text* sebesar 216 bit membutuhkan memori 240 bita. Proses enkripsi *plain text* sebesar 352 bit membutuhkan memori 384 bita.

6.4 Pengujian Keamanan

Pengujian keamanan dilakukan untuk memastikan sistem telah memenuhi mekanisme keamanan *CIA (Confidentiality*, *Integrity*, *Availability)* (Farooq, 2015). Penelitian ini berfokus pada unsur *confidentiality* atau kerahasiaan data yang dikirim melalui jaringan nirkabel.

6.4.1 Kerahasiaan Data Saat Proses Pengiriman

Kasus uji kerahasiaan data saat proses pengiriman melalui jaringan *wifi* dapat dilihat pada Tabel 6.28.

Tabel 6.28. Pengujian unsur kerahasian data.

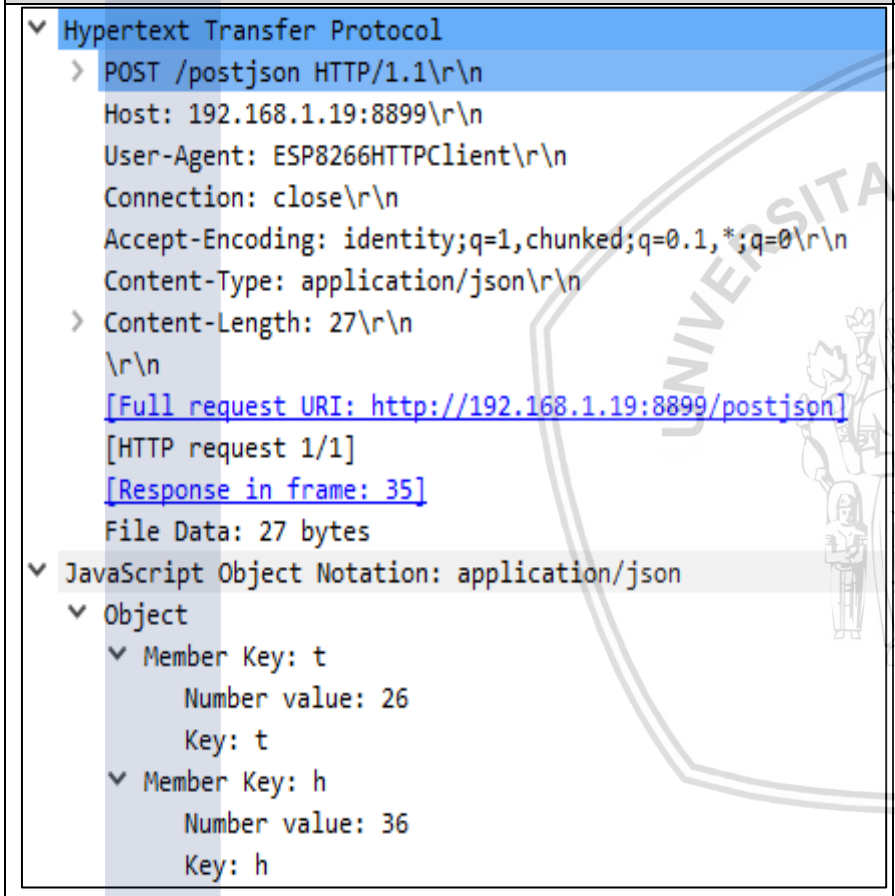
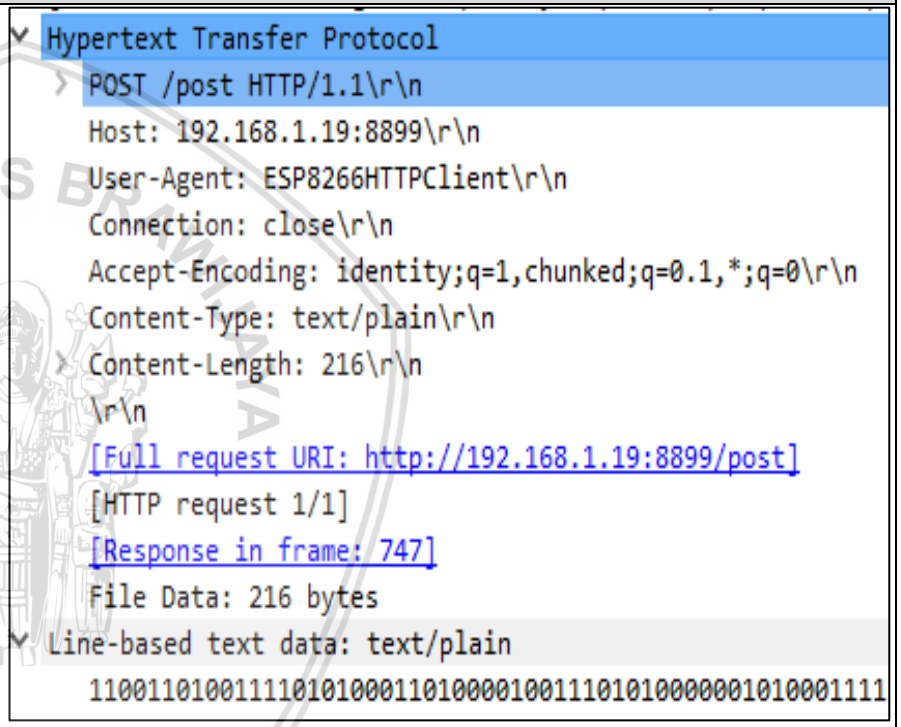
Kode	PS-01
Nama uji	Pengujian kerahasiaan data saat proses pengiriman
Tujuan pengujian	Melihat dan memastikan kerahasiaan data yang dikirim oleh klien menuju ke server melalui jaringan nirkabel.

Tabel 6.29. (Lanjutan) Pengujian unsur kerahasian data.

Prosedur pengujian	1. Melihat hasil tangkapan paket pada proses pengiriman data dari klien menuju server REST menggunakan aplikasi Wireshark. 2. Membandingkan penangkapan paket saat klien mengirim *plain text* dengan saat mengirim *cipher text*.

Setelah prosedur pengujian selesai dijalankan, ditemukan bahwa data *plain text* yang dikirim oleh klien dapat terbaca dan dimengerti dengan baik oleh manusia. Sedangkan data dalam bentuk *cipher text* yang dikirim oleh klien tidak dapat dimengerti oleh manusia. Hasil pengujian tersebut dapat dilihat pada Tabel 6.30 di halaman selanjutnya.

Tabel 6.30. Hasil pengujian kerahasiaan data menggunakan aplikasi Wireshark.

Paket pengiriman *plain text*	Paket pengiriman *cipher text*
	
Gambar 6.18. Hasil tangkapan paket pengiriman *plain text* pada Wireshark.	**Gambar 6.19. Hasil tangkapan paket pengiriman *cipher text* pada Wireshark.**

BAB 7 PENUTUP

Bab penutup berisi tentang kesimpulan dan saran yang diperoleh berdasarkan hasil dari proses perancangan, implementasi, dan pengujian.

7.1 Kesimpulan

Berdasarkan hasil proses perancangan, implementasi, dan pengujian yang telah diselesaikan sebelumnya, didapatkan beberapa kesimpulan sebagai berikut:

1. Hasil pengujian fungsionalitas dan keamanan menunjukkan bahwa algoritme Lizard dapat diimpelementasikan untuk memenuhi unsur kerahasiaan dalam proses pengiriman data dari perangkat NodeMCU menuju server basis data.
2. Berdasarkan pengujian validasi *keystream* algoritme Lizard, 128 bit *keystream* yang dibentuk oleh mikrokontroler NodeMCU memiliki hasil yang sesuai dengan *test vector* algoritme Lizard. Oleh karena itu, dapat dikatakan bahwa algoritme Lizard yang telah diimplementasi bersifat valid dan benar.
3. Proses pembentukan *keystream* algoritme Lizard sebesar 216 dan 352 bit pada perangkat mikrokontroler NodeMCU membutuhkan waktu $\pm$ 0,01 detik. Memori yang dibutuhkan dalam proses pembentukan 216 dan 352 bit *keystream* pada mikrokontroler NodeMCU adalah sebesar 2720 dan 2864 bita atau 3,4% dari memori total perangkat. Hasil pengujian juga menunjukkan bahwa variasi kunci dan *IV* algoritme Lizard mempengaruhi penggunaan waktu yang dibutuhkan oleh perangkat mikrokontroler dalam proses pembentukan *keystream,* namun tidak mempengaruhi penggunaan memori.
4. Proses enkripsi *plain text* sebesar 216 bit membutuhkan waktu 0,0282 detik dan memori sebesar 240 bita. Sedangkan untuk proses enkripsi *plain text* sebesar 352 bit perangkat mikrokontroler NodeMCU membutuhkan waktu 0,0493 detik dan memori sebesar 384 bita atau 0,3% dari memori total. Semakin besar *plain text*, maka semakin besar waktu dan memori yang dibutuhkan perangkat NodeMCU dalam menjalankan proses enkripsi.

7.2 Saran

Berdasarkan kesimpulan penelitian yang telah diuraikan sebelumnya, direkomendasikan beberapa saran sebagai berikut:

1. Penelitian berikutnya dapat dilakukan dengan mengimplementasikan algoritme integritas untuk memenuhi unsur keamanan *CIA* (*Confidentiality*, *Integrity*, *Availability*) pada sistem.
2. Mengimplementasikan jenis algoritme kriptografi *block cipher* atau algoritme *stream cipher* yang lebih baru dari Lizard pada perangkat NodeMCU.

DAFTAR PUSTAKA

Ekklesia, D., 2016. *Studi dan Implementasi Pengamanan basis data dengan Teknik Kriptografi Stream Cipher*. Institut Teknologi Bandung.

Farooq, M.U., Waseem, M., Khairi, A., & Mazhar, S., 2015. *A Critical Analysis on the Security Concerns of Internet of Things (*IoT*)*. International Journal of Computer Applications (0975 8887) vol. 111 – No. 7.

Hamann, M., Krause, M., & Meier, W., 2017. *LIZARD – A Lightweight Stream Cipher for Power-constrained Devices*. Germany: University of Mannheim.

Laine, M., 2011, REST*ful Web Services for the Internet of Things*. Finlandia: Aalto University School of Science.

Mukhlasin, H. *Yii2: Authentication on Yii2* REST*ful Web Service*. 8 Desember 2015. http://www.hafidmukhlasin.com/2015/12/08/yii2-authentication-on-yii2-web-service-restful-part-1/, diakses pada 16 Desember 2017.

Nurrohmah, A., Kusyanti, A., & Primananda, R., 2017. I*mplementasi Algoritme Grain V1 Dan 128 Bit Pada Arduino Mega 2560*. S1. Universitas Brawijaya: Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer Vol. 2, No. 4, April 2017, hlm. 1436-1445.

Richardson, L., & Ruby, S. 2007. REST*ful Web Services*. United States of America: O'Reilly Media.

Rose, K., 2015. The Internet Of Things: An Overview. [pdf] Internet Society (ISOC). Tersedia di: <https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf> [Diakses 4 Januari 2018]

Roman, R., Najera, P., & Lopez, J., 2011. *Securing the Internet of Things*. Spanyol: University of Malaga.

Sembiring, I., & Asang, M., 2017. *Keamanan Data Pada Perangkat Internet Of Things Menggunakan Metode Public-Key Cryptography*. *Faculty of Information Technology* Universitas Kristen Satya Wacana: Jurnal AITI Vol.14 no 1 tahun 2017, hal. 80-87.

Vandikas, V. Dan Tsiatsis, V., 2016. *Microservices In* IoT *Clouds*. Cloudification of the Internet of Things (CIoT): IEEE.